VAD360: Viewport Aware Dynamic 360-Degree Video Frame Tiling

Chamara Kattadige
ckat9988@uni.sydney.edu.au
The University of Sydney
Sidney, Australia

Kanchana Thilakarathna
kanchana.thilakarathna@sydney.edu.au
The University of Sydney
Sydney, Australia

ABSTRACT

360° videos a.k.a. spherical videos are getting popular among users nevertheless, omnidirectional view of these videos demands high bandwidth and processing power at the end devices. Recently proposed viewport aware streaming mechanisms can reduce the amount of data transmitted by streaming a limited portion of the frame covering the current user viewport (VP). However, they still suffer from sending a high amount of redundant data, as the fixed tile mechanisms can not provide a finer granularity to the user VP. Though, making the tiles smaller can provide a finer granularity for user viewport, high encoding overhead incurred. To overcome this trade-off, in this paper, we present a computational geometric approach based adaptive tiling mechanism named VAD360, which takes visual attention information on the 360° video frame as the input and provide a suitable non-overlapping variable size tile cover on the frame. Experimental results shows that VAD360 can save up to 31.1% of pixel redundancy before compression and 35.4% of bandwidth saving compared to recently proposed fixed tile configurations, providing tile schemes within 0.98(±0.11)s time frame.

CCS CONCEPTS

• **Human-centered computing → Virtual reality**.

KEYWORDS

360°/VR video streaming, Video frame tiling

1 INTRODUCTION

Over the past decade, video streaming has been dominating the global internet traffic accounting for 80% of total traffic [20]. Among them, 360° vidoes, a.k.a. spherical videos, is becoming increasingly popular as it enables immersive streaming experience. Along with the advancement of Head Mount Devices (HMDs) (e.g., Facebook Oculus [1], Microsoft Hololens [3]) and smartphones, content providers (e.g., YouTube (YT) [4] and Facebook (FB) [2]) have already facilitated both on-demand and live-streaming of 360° videos.

Despite the gaining popularity of 360° videos, its intrinsic larger spherical view has posed several challenges against providing expected high user Quality of Experience (QoE). Firstly, 360° videos demand high network bandwidth since spherical video frames should be 4–6 times larger than normal video frames to achieve the same user perceived quality [19]. Secondly, processing such video frames at end devices imposes high resource utilization is problematic particularly with the limited resources such as in smartphones. Finally, 360° video streaming requires strictly low latency, which should be within the *motion-to-photon delay*[1] (just above 25ms), otherwise the user may suffer from severe motion or cyber-sickness [13].

Recently proposed viewport (VP) –current visbile area, typically encompass 100°×100° Field of View (FoV)– adaptive streaming [27], which streams only a selected portion of the 360° video frame, has shown a great promise in addressing the above issues. Especially, the tiling mechanism, which divides the entire frame into tiles and send only the tiles within the user VP can reduce the amount of data transferred to the client and increases the user perceived video quality compared to streaming entire video frame [10, 19, 28, 29]. However, due to the fix number of tiles (typically ranges between (24–36)) [10, 19] and their fixed sizes, optimum gain of bandwidth saving is not achieved by these mechanisms. On the one hand, fixed tiling schemes fails to provide finer boundary to the user FoV, therefore, there is a high pixel redundancy [27]. On the other hand, they do not have awareness of visually attractive regions and FoV distortion on Equirectangular Projection (ERP) when encoding the tiles. For example, corresponding regions of the polar regions of the spherical view on the ERP frame are highly distorted and have less viewing probability. However, fixed tiling encodes these regions at the same bit-rate levels and in smaller tile size as in the equatorial regions, adding unnecessary quality overhead and losing many compression opportunities compared to having bigger tiles [11, 27]

To this end, provisioning an viewport aware adaptive tiling mechanism with variable sized tiles can provide fine granularity for the FoV and also increase compression gain. This enables content providers to reduce encoding overhead incur by tiling the videos and identify the tiles which should be in high quality at a fine granularity. At the network provider side, transmitted data volume will be further decreased reducing the bandwidth consumption for 360° video streaming whilst providing opportunities to re-innovate caching mechanisms for 360° videos. Finally, existing DASH (Dynamic Adaptive Streaming over HTTP) protocols on 360° videos can be enhanced to provide better user QoE at the client side.

Achieving such an adaptive tiling mechanism is challenging due to sheer volume of videos need to be processed, stored and streamed from the content providers perspective. In on aspect, running algorithms to generate suitable tile schemes should not demand high processing power, as the servers are already in high utilization to support excessive demand of current video streaming services. In another aspect, algorithms itself should process in a minimal time period. However, recently proposed dynamic tiling solutions [18, 27, 31] compromise both these aspects by encoding all possible tile schemes for a given frame even in more than the quality levels provided in DASH protocols [18]. Furthermore, Integer

[1]Delay between user head movement and the finishing rendering the corresponding VR frame on to the display

Liner Programming (ILP) based solutions and exhaustive searching mechanisms on all possible solutions provided in these proposals requires a significant processing time.

In this paper, we propose VAD360 (**V**iewport **A**daptive **S**calable 360-degree Video Frame **Til(e)**ing), an adaptive tiling mechanism for 360° video frames supported by dynamic nature of user VPs in 360° video streaming. In VAD360, we leverage a computational geometric approach, which we call *Minimal Non-overlapping Cover (MNC)* algorithm [23, 24], to devise a suitable tile scheme by partitioning rectilinear polygons generated by combining basic tiles from 10 × 20 grid overlaid on the 360° video frame. To generate the rectilinear polygons, VAD360 consists with semi-automated thresholding mechanism, which divides the 360° video frame into multiple sub regions based on the visual attraction level, i.e. saliency, of pixels of the frame. Moreover, taking FoV distortions on the ERP frame and removing potential overlaps, VAD360 further reduces the downloaded data volume, transmitted pixel redundancy and processing time for end-to-end tile generation process.

We leverage 30, 360° videos in 1 min duration with 30 user VP traces in each to build VAD360 and validate its performance. Our experimental results shows that VAD360 can save up to 31.1% of pixel redundancy before compression and 35.4% of bandwidth saving compared to recently proposed fixed tile configurations (cfgs.). Moreover, circumventing the time consuming process of encoding all possible tile combinations for exhaustive/ILP based searching algorithms, VAD360 is able to generate suitable tile scheme with in avg. of 0.98 (±0.11)s processing time with at least 80% of individual user VP coverage in high quality. Artficats of the work are–*hidden for double blind submission*–.

2 RELATED WORK

VP-aware 360° **video streaming:** Plethora of works have been done in VP-aware 360° video streaming optimization [5, 10, 19, 21, 28, 29]. In these mechanisms, a predicted user VP is sent to the content servers and a selected portion of the frame covering the requested VP is transmitted to the client. The most prominent way of selecting the required portion is first, dividing the entire frame into fixed number of tiles and select only the tiles fall with in the user VP [10, 19, 28, 29]. The overarching goal of VP aware streaming is to reduce the amount of data transmitted through the network and increase the quality of the streamed portion to increase the user perceived QoE. A major drawback of these proposals is that tiles in fixed size transmit high amount of redundant data as they can not provide finer boundary to the user FoV. Although, smaller size tiles can create a finer boundary, it increases the encoding overhead which results in higher bandwidth consumption [11, 27].

Adaptive tiling schemes on 360° **video frames:** In contrast to the uniform size tiling, variable size tile schemes are proposed in [12, 30]. They divide the frame into fixed num. of tiles but vary their size according to the latitude. Diversely, combining set of basic tiles in fixed minimum size to form larger tiles in variable amount and size are presented in [18, 27, 31]. Both [27] and [31] leverage ILP based solution to find the best tile cfgs. taking the server side storage size and streaming data volume as the cost functions. Ozcinar *et al.* present a exhaustive searching method to derive tiles while allocating a given bitrate budget [18]. These approaches are lack of scalability due to two reasons. First, they require to encode all possible tile schemes, which may exceed 30000 of solutions [27] incurring high encoding time. Second, algorithms such as exhaustive searching/ILP itself need longer processing time.

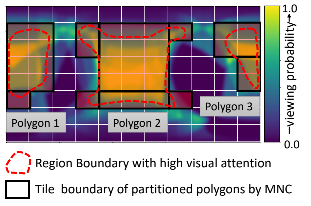
In contrast to the above method, we propose a scalable, adaptive tiling mechanism leveraging a computational geometric approach, which can provide high quality tiles in user VP, whilst reducing the pixel redundancy before compression and streamed volume of data.

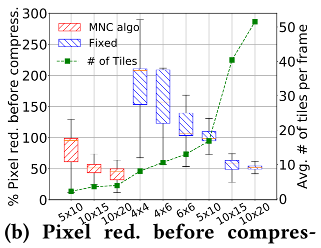
3 BACKGROUND AND MOTIVATION

The overarching goal in VAD360 is to partition video frames leveraging visual attention level of each pixel in the frame as shown in Fig. 1a. Generating visual attention maps has been well studied in the literature [15, 17, 18, 27], which are often developed by either analysing content features [15, 17] or/and by analysing past VP traces of the video [17, 18, 27]. The primary challenge addressed in this work is how we find optimal partitioning of frames given a visual attention map. For this purpose, VAD360 leverages computational geometric approach proposed in [24, 26].

We now presents basics of the approach and its applicability for 360° video frame partitioning. Moreover, our preliminary experiments on individual user viewports shows how efficient this utilized algorithm in frame partitioning compared to the fixed tile configurations in terms of % pixel redundancy before compression.

3.0.1 MNC algorithm. The original rectilinear polygon partitioning algorithm proposed in [24, 26] creates a rectangular tile cfg. covering the polygon region with fewest number of tiles in variable size, a.k.a. *Minimal Overlapping Cover (MNC)*. In breif basic *MNC* algorithm runs on hole free rectilinear polygons first taking the concave (and convex) vertices in polygon boundary to find maximum independent chords. These independent chords divide the polygon into sub-rectilinear polygons which can be further partitioned by drawing chords (vertically/horizontally) from the concave vertices, from which a chord has not been drawn yet. We refer interesting users to [24, 26] for detail implementation of *MNC* algorithm.


Polygon 1
Polygon 2
Polygon 3
viewing probability
1.0
0.0
Region Boundary with high visual attention
Tile boundary of partitioned polygons by MNC
(a) Partitioning regions with high visual attention using basic *MNC* algorithm



(b) Pixel red. before compression & no. of tiles per frame

Figure 1: *MNC* based partitioning and Comparison with fixed tile cfgs.

3.0.2 Applicability of MNC algorithm for partitioning. Fig. 1a shows that on the visual attention maps generated using past VP traces, we can create rectilinear polygons surrounding the regions with high VP probability. These polygons, which are comprised of set of basic tiles (i.e., smallest tile that can not be be further partitioned denoted as BT) can be partitioned by the *MNC* algorithm as in Section. 3.0.1. However, *MNC* algorithm is unaware of visual attention as information of vertices of polygon is enough for the partitioning. Leveraging simple pre/post-processing steps on the video frame, we can inject viewport awareness to the *MNC* algorithm converting the process for quality adaptive tiling. which is further elaborated in Section 4.1.

3.0.3 Comparison with fixed tile configuration. We run *MNC* algorithm on individual user viewports[2] and compare the generated tile schemes with fixed tile cfgs. [10, 14, 19] measuring the % pixel redundancy before compression. We leverage VP maps of randomly selected 5 users from 5 sample videos representing the categorization provided in [7][3]. We consider three basic tile configurations (5×10, 10×15 and 10×20) to generate rectilinear polygons as the input to the *MNC* algorithm. We compare the derived tile schemes from *MNC* algorithm with five fixed tile cfgs. (4×6, 6×6, 5×10, 10×15 and 10×20) applied on the same user viewport. We measure % pixel redundancy before compression as in Eq. 1.

$$Pixel\ red.\ before\ compression = \frac{N_T - N_{FoV}}{N_{FoV}}\% \qquad (1)$$

where N_T and N_{FoV} represent number of non-zero pixels in tiles which overlaps with the user FoV and considered FoV region respectively. Fig. 1b shows the analysis results. We see that 10×20 cfg. results in lowest % pixel redundancy for both *MNC* partitioning and fixed tile approach. However, to cover the same single user VP, 10×20 cfg. in fixed tiling requires $\times 25$ the no. of tiles generated by *MNC* algorithm. Therefore, compared to *MNC* algorithm , more encoding overhead incur on fixed tile cfgs. as the tile sizes are smaller and higher in amount [11, 27].

4 DESIGN OF VAD360 FRAMEWORK

Now, we present VAD360 holistic architecture, which includes *Pre-processing*, *Partitioning* and *Post-processing* of 360° video frames as illustrated in Fig. 2.

4.1 VAD360 overview

The objective of the *Pre-processing* step is to identify different regions in a frame, according to the expected visual attention in order to inject viewport awareness to the *MNC* algorithm and reduce the complexity of detected polygons for partitioning. We first create averaged *Viewport Map (VM)* combining individual user viewport maps and then apply a hierarchical thresholding mechanism to detect visual attention blobs[4] for the following four regions, as depicted in Fig. 3. The first two regions are defined on the area which covers at least 80% of the user VPs namely, $R(FoV_f)$ which covers the most attractive regions and $R(FoV)$ covering the remaining area. We define $R(Buf)$ as an additional buffer region to cover VPs slightly deviated from the $R(FoV_f)$ and $R(FoV)$ which is likely to be outside the region covering at least 80% of user VPs. Finally, $R(OoV)$ is the remaining area of VM with the lowest viewing probability. Details of *Pre-processing* steps are presented in Section 4.2.

In *Partitioning* step, we run *MNC* algorithm on the above four regions separately. We create rectilinear polygon boundary around the blobs in these regions and partition them into non-overlapping tiles denoted as DT (Derived Tile) comprised of set of BTs (Basic Tiles). Since we consider multiple blobs in VM frame separately, there can be overlaps between the derived tiles, if the considered blobs are closed to each other. Therefore we remove such overlaps during $R(FoV)$ and $R(Buf)$ partitioning. Details of *Partitioning* steps are presented in Section 4.3.

In *Post-processing* step, we further split DTs which are larger than FoV ($100^\circ \times 100^\circ$) considering its distortion when projecting on to the Equirectangular (ERP) Frame. Finally, bit rate can be allocated to each DT considering the tile properties such as average pixel intensity, tile size and tile location. Details of partitioning steps are presented in Section 4.4.

Here on-wards, we denote a given user and video by i and j respectively, and no. of users in a video as u_j. Pixel coordinates of video frames are denoted as (m,n), where $0 \le m < H$, $0 \le n < W$.

4.2 Pre-processing

We now describe the frame pre-processing pipeline identify salient regions to be partitioned by the MNC algorithm as the output.

4.2.1 Frame sampling (Fig. 2-a). Frame rate for videos can vary between 24-60fps while 30fps is common in general. It is not necessary to partition each and every frame in a video because; (i) it is safe to assume that FoV of the user is fixed at a certain position for certain period [17], (ii) having different tile scheme for each frame can reduce compression gain in encoding, and (iii) based on the above two facts, running a partitioning algorithm on every frame adds unnecessary computational cost.

To decide the most suitable frame gap, we analyse the relationship between the temporal and spatial user viewport behavior in light of the peripheral vision of human eye. Fig. 4 shows the angular difference of yaw and pitch distribution against the sampling frame gaps from 0.1–2.0s by every 0.1s. Human vision can perceive high quality only at *near peripheral region*, i.e. within 30° range, [6, 8, 22]. Based on that fact, we make a fair assumption that from a fixation point, the user can view with almost the same visual quality at maximum of 30° without changing the fixation point. According to practical VP traces, up to 0.8s of frame gap can tolerate 30° angle difference in Yaw direction. Including a safety margin of 5°, we decide 0.5s as the frame gap to refresh the tile scheme without making a significant harm to the user perceived quality. Compared to 1s tile scheme refreshing period found in literature [18, 27], which can leads ($\geq$ 35°) angle difference, 0.5s gap can better adapt to FoV changes. Also, it reduces the encoding overhead and time for processing if we are to consider every single frame for partitioning.

4.2.2 Viewport Map (VM) generation (Fig. 2-b). Despite many different approaches for generating visual attention maps, we leverage historical user VP traces to generate VMs. [18] claims that 17 users are sufficient to create representative VP map. Therefore, we consider 20 users from each video to generate our VM frames. First, given the centre of VP: c_i, of each user i in $< yaw, pitch >$ angles, we create a binary map, V_i according to Eq. 2.

$$V_i(m,n) = \begin{cases} 1, & \text{if } (m,n) \in F_i \\ 0, & \text{otherwise} \end{cases} \qquad (2)$$

We assume a $100^\circ \times 100^\circ$ FoV area (F_i) representing the FoV of the majority of commercially available HMDs [7]. We also project spherical coordinates of pixels (x, y) to ERP format (m, n) considering the geometrical distortion, creating more dispersed pixel distribution towards the upper and bottom parts of the frame (i.e.,

[2]Unlike the visual attention maps, which combines multiple user VPs, an individual user VP is a binary map representing FoV by 1 and outside the FoV by 0.

[3][7] put videos in into 5 categories: riding, moving focus, exploration, rides and miscellaneous (a combination of previous types)

[4]Regions with (near)concentric user VPs

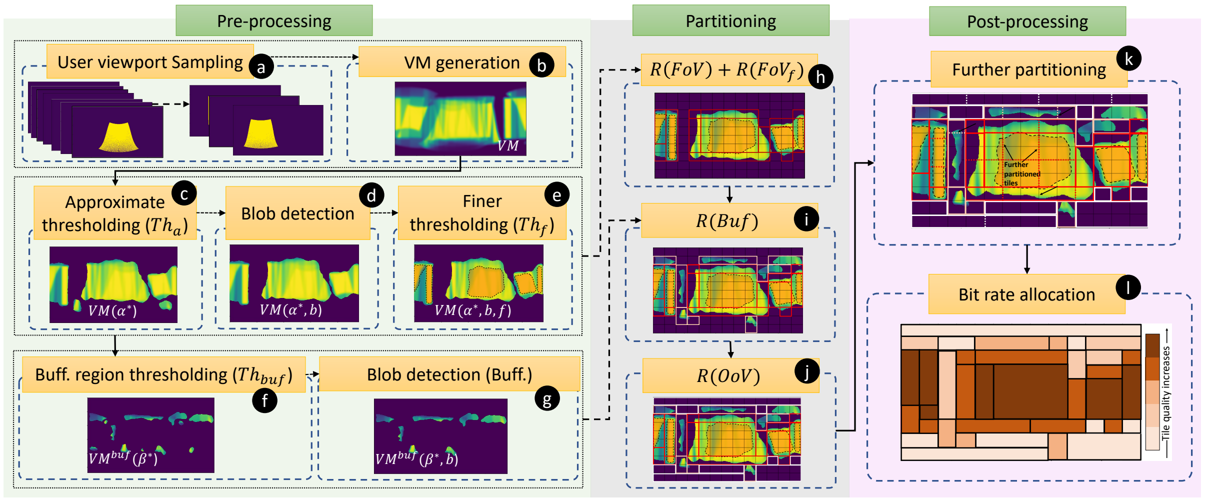


Figure 2: Overall architecture of VAD360 including Pre-processing, Partitioning and Post-processing steps.

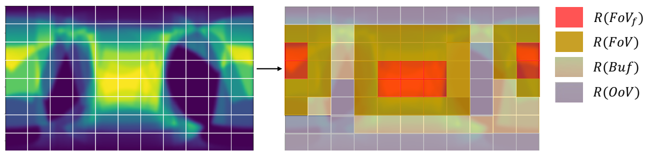


Figure 3: Four FoV regions considered in frame partitioning.

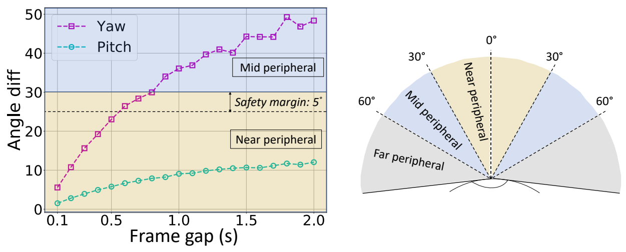
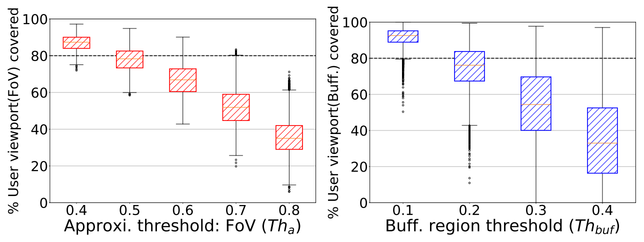


Figure 4: (left): Angle diff. vs the frame gap. (right): Peripheral vision of human eye



(a) Th_a validation (b) $Th_{buff.}$ validation

Figure 5: % user viewport distribution on thresholded area under different threshold values for both approximate (Th_a) and buffer (Th_{buff}) region thresholding

corresponding to the polar region of the spherical frame). We then generate VM taking the average of all users, u_j. We generate normalized avg. Viewport Map, VM^{norm}, after histogram equalizing the VM and dividing by maximum pixel value (=255) as in Eq. 3

$$VM^{norm} = \frac{1}{255} HIST(VM) \quad (3)$$

4.2.3 Approximate Thresholding (Fig. 2-c). The objective of this step is to filter $R(FoV_f)$+$R(FoV)$ regions representing at least 80% of the user VPs. To determine the appropriate threshold of VM^{norm}, we first calculated the overlap between individual user VPs and the thresholded region for a range of thresholds, Th_a as shown in Fig. 5a. The results show that except 0.8, all other values can cover at least 80% of FoV region at least for one frame. We also note that 80% margin covers VPs of at least 17 users, out of 20 users, which is claimed to be the minimum number of user VPs needs to generate representative VM [18]. By this analysis, we reduce our search space for appropriate threshold Th_a between 0.4 to 0.7. Note that higher the Th_a, smoother the thresholded region boundary reducing the complexity of partitioning algorithm.

We present Algorithm 1 to select highest possible threshold for each frame, Th_a. First, we create binary map $VM(\alpha)$, by thresholding the VM^{norm} using $Th_a = \alpha$ (line 12). Then we measure the % of FoV overlap of each user VP (V_i) with $VM(\alpha)$ calculating an average value, $s_{avg}(\alpha)$ (line 13-17). Finally, if the $s_{avg}(\alpha)$ is no more giving 80% coverage we stop further process and assign Th_a with the previous α value. If none of the Th_a satisfy 80% FoV coverage, we select $Th_a = 0.4$ (line 18-24). We apply $Th_a = \alpha^*$ on VM^{norm} and denote the resulting frame with $R(FoV_f)$+$R(FoV)$ as $VM(\alpha^*)$.

4.2.4 Blob detection (Fig. 2-d). Due to the non-uniform dispersion of the user VP, VM can contain multiple blobs. Aim of this step is to identify these blobs and exclude non-significant small blobs reducing the complexity of partitioning process. Without loss of generality, we select the blobs in $VM(\alpha^*)$, covering at least 95% of $R(FoV)$+$R(FoV_f)$. Algorithm 2 summarizes the blob selection process given the $VM(\alpha^*)$ as the input. Firstly, the function $G(VM(\alpha^*))$ outputs all the blobs in $VM(\alpha^*)$ frame as a set, B, followed by sorting in descending order according to the blob size (line 10-11). After that, we cumulatively sum up the blob size, starting from the largest one and stop the process when the total selected blobs size (Z_{sel}) exceeds 95% of total thresholded area ($Z_{VM(\alpha^*)}$) in $VM(\alpha^*)$ (line 12-18). Finally, a map, $VM(\alpha^*, b)$, is created combining all the selected blobs using the function $H(B^{sel})$ (line 19).

Algorithm 1 Determine Th_a

```
Input
    $VM^{norm}$   Normalized VP map
    $\{V_i\}$   Set of individual user VP maps $\forall i \in [1, u_j]$
Vaiable
    $\alpha$   a given $Th_a$ value s.t. $\alpha \in \{0.4, 0.5, 0.6, 0.7\}$
    $VM(\alpha)$   binary map of $VM^{norm}$ after thresholding by $\alpha$
    $I(\alpha)$   Intersection map between $V_i$ and $VM(\alpha)$
    $s_i(\alpha)$   % overlap between $V_i$ and $I(\alpha)$
    $S(\alpha)$   a set containing $s_i(\alpha)\ \forall i \in [1, u_j]$
    $s_{avg}(\alpha)$   avg. of $s_i(\alpha)$ for all users $u_j$
for $\alpha$ = 0.4 to 0.7, step = 0.1 do
    $VM(\alpha) = \begin{cases} 1, & \text{if } VM^{norm}(m,n) \geq \alpha \\ 0, & \text{otherwise} \end{cases}$
    for $i = 1$ to $u_j$ do
        $I_i(\alpha) = VM(\alpha) \cap V_i$   ▷ get intersection map
        $s_i(\alpha) = \frac{\sum\sum I_i(\alpha)}{\sum\sum V_i}\%\ \ \forall m \in [0, H), \forall n \in [0, W),$
        $S(\alpha).add(s_i(\alpha))$   ▷ store the % overlap user $i$
    $s_{avg}(\alpha) = \frac{1}{u_j}\sum_{i=1}^{u_j} s_i(\alpha),$ s.t. $s_i(\alpha) \in S(\alpha)$
    if $s_{avg}(\alpha) < 80\%$ then   ▷ check for 80% coverage
        if $\alpha > 0.4$ then
            $Th_a^* = \alpha - 0.1$
        else   ▷ if none of the $Th_a$ satisfy 80% coverage
            $Th_a = 0.4$
        return $Th_a$
$Th_a = 0.7$   ▷ even if $\alpha = 0.7$ satisfy the 80% coverage
return $Th_a$
```

Algorithm 2 Select blobs from $VM(\alpha^*)$

```
Input
    $VM(\alpha^*)$   Approximate thresholded frame
Vaiable
    $b_l$   $l^{th}$ blob in $VM(\alpha^*)$, $l \in [1, l_{max}]$, $l_{max}$: maximum no. of blobs in $VM(\alpha^*)$
    $z_l$   size of $l^{th}$ blob
    $B$   set containing the all the blobs from $VM(\alpha^*)$
    $B_{sel}$   Set containing the selected blobs
    $Z_{sel}$   total size of a selected blobs from $B_{sel}$
    $Z_{VM(\alpha^*)}$   Total thresholded area of $VM(\alpha^*)$
$B \leftarrow G(VM(\alpha^*))$   ▷ get all the blobs to set $B$
$B^{sort} \leftarrow sort(B)$ : (descending order of blob size)
$Z_{sel} = 0$
while $l < l_{max}$ do
    $B^{sel}.add(b_l)$   ▷ Add blobs to a set
    $Z_{sel} = Z_{sel} + z_l$   ▷ add blob size cumulatively
    if $(Z_{sel}/Z_{VM(\alpha^*)})\% \geq 95$ then   ▷ check for 95% coverage
        break
    $l = l + 1$   ▷ increment the blob count
$VM^{\alpha^*,b} \leftarrow H(B^{sel})$   ▷ create pixel map from selected blobs
return $VM^{\alpha^*,b}$
```

4.2.5 Finer thresholding (Fig. 2-e). To provide a higher quality for the most attractive region, in this step, we filter $R(FoV_f)$ from $VM(\alpha^*, b)$, defining *Finer threshold*, Th_f. Without loss of generality we set $Th_f = 0.9$ to identify the region $R(FoV_f)$ boundaries. We expand this boundary to generate perfect rectangular polygon as we discussed in Section 4.3. We denote the finer thresholded frame as $VM(\alpha^*, b, f)$, which is the input for *MNC* algorithm for $R(FoV)+R(FoV_f)$ partitioning .

4.2.6 Buffer region thresholding and blob detection (Fig. 2-f and 2-g). The objective of this step is to filter $R(buf)$ which covers slight variations of user VPs. We extract $R(Buf)$ from the area not covered by the Derived Tiles (DT) from $VM(\alpha^*, b, f)$ on the initial VM which is denoted as VM^{buf} (Eq. 4). We use the same DT information to obtain corresponding buffer regions in V_i (individual user viewports) namely, V_i^{buf}, as in Eq. 5.

$$VM^{buf} \leftarrow VM \cap (VM^T(\alpha^*, b, f))' \tag{4}$$

$$V_i^{buf} \leftarrow V_i \cap (VM^T(\alpha^*, b, f))'\ \forall i \in [1, u_j] \tag{5}$$

where $VM^T(\alpha^*, b, f)$ and $(VM^T(\alpha^*, b, f))'$ denotes the DT overlay on the $VM(\alpha^*, b, f)$ by *MNC* algorithm and the remaining region on the frame respectively.

In order to extract a suitable buffer region from VM^{buf}, we apply *Buffer threshold* (Th_{buf}) and *Blob detection* as the same way we followed in Th_a finding and *Blob detection* in approximate thresholding (cf. Section 4.2.3 and 4.2.4). Hence, we compute % user viewport (V_i^{buf}) covered by the thresholded region from VM^{buf}, for the $Th_{buf} \in \{0.1, 0.2, 0.3, 0.4\}$. Fig. 5b shows that all threshold values can provide ($\geq 80\%$)[5] buffer viewport coverage, therefore, we dynamically select Th_{buf} value from the above set. We apply Algorithm. 1 simply changing the threshold values (line-5) and replacing VM^{norm} (line-2 & 12) and V_i (line-3 & 14) with VM^{buf} and V_i^{buf} respectively. We define the best $Th_{buf} = \beta^*$ and thresholded buffer frame as $VM_i^{buf}(\beta^*)$. After that to exclude the non-significant smaller blob region, we apply Algorithm 2 on $VM^{buf}(\beta^*)$, by simply replacing $VM(\alpha^*)$ with $VM^{buf}(\beta^*)$ (line 2 & 10). We denote blob filtered buffer frame as $VM^{buf}(\beta^*, b)$.

4.2.7 OoV extraction. The goal of this step is to extract $R(OoV)$ to derive low quality DT to satisfy any anomaly user VP. We filter out $R(OoV)$ removing the area covered by DT overlay on $VM(\alpha^*, b, f)+VM_i^{buf}(\beta^*, b)$ area (similar to Eq. 4). No further pre-processing is applied to OoV region as no significant pixel value distribution is observed. We denote the OoV region as VM^{oov}.

4.3 Partitioning

VAD360 frame partitioning step runs the *MNC* algorithm on $R(FoV_f)$+$R(FoV)$, $R(Buf)$ and $R(OoV)$ regions separately to generate DTs. We start with creating a rectilinear polygon covering each blob followed by running basic *MNC* algorithm in Section 3.0.3.

Firstly, for $R(FoV)$ and $R(FoV_f)$ partitioning (Fig. 2-h), we leverage $VM(\alpha^*, b, f)$. Fig. 6 shows $(R(FoV) + R(FoV_f))$ partitioning process. We expand the detected polygon in $R(FoV_f)$ (i.e., polygons in blue color) converting to a perfect rectangle. The boundary is extended to the minimum and maximum (m,n) locations as long

[5]Covering corresponding V_i^{buf} from at least 17 users out of 20

as it does not exceed the $R(FoV)$ polygon boundary as shown in red color arrows. By this step, we make the partitioning process simpler and create an extra buffer for $R(FoV_f)$s to be encoded at higher quality. Fig. 6 (b) shows that polygons for $R(FoV)$ (e.g.,**R3**) is extracted removing all the polygons generated for $R(FoV_f)$ (e.g., **R1**, **R2**). Note that, extracting $R(FoV_f)$ as rectangles create holes in $R(FoV)$ region. Since, the basic *MNC* algorithm is proposed for hole-free rectilinear polygons, we have added additional steps on top of VAD360 *MNC* implementation. In brief, when finding maximum independent chords, we take vertices of holes into account. Then, from the remaining vertices which have not connected with any independent chord, we draw extra chords to complete the partitioning. We avoid hovering any chord on the holes.

Secondly, taking $VM^{buf}(\beta^*, b)$ and VM^{oov} frames *MNC* algorithm partitions $R(Buf)$ and $R(OoV)$ respectively (Fig. 2-i and 2-j) without any further processing on the polygons around the blobs detected. Finally, we see that, due to the close proximity of selected blobs certain DTs may overlap on each other. Given such two tiles, we remove the overlapped region only from the smaller DT ensuring the non-overlap DT coverage on the entire frame.

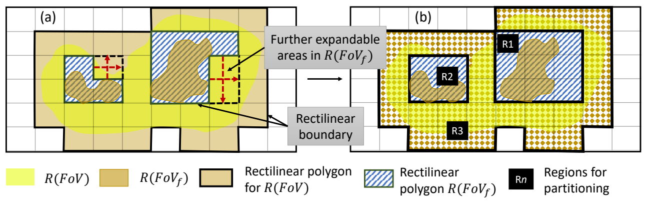

(a)
(b)
Further expandable areas in $R(FoV_f)$
Rectilinear boundary
R1
R2
R3
🟨 $R(FoV)$
🟫 $R(FoV_f)$
Rectilinear polygon for $R(FoV)$
▨ Rectilinear polygon $R(FoV_f)$
Rn Regions for partitioning

Figure 6: $R(FoV)+R(FoV_f)$ partitioning: (a)-before & (b)-after expanding rectilinear polygon of $R(FoV_f)$

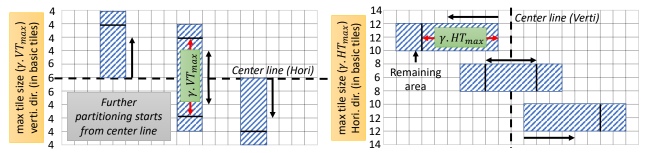

max tile size ($\gamma.VT_{max}$) verti. dir. (in basic tiles)
4
4
4
4
6
6
6
4
4
4
4
$\gamma.VT_{max}$
Center line (Hori)
Further partitioning starts from center line
max tile size ($\gamma.HT_{max}$) Hori. dir. (in basic tiles)
14
12
12
10
8
6
8
10
12
12
14
$\gamma.HT_{max}$
Center line (Verti)
Remaining area

Figure 7: Further partitioning mechanism and maximum allowable tile size in horizontal and vertical direction based on the vertical position of center of a given tile.

4.4 Post-processing

4.4.1 Further partitioning of bigger DTs. We further partition DTs beyond certain limit of the size, in order to reduce the pixel redundancy. Since, we consider polygon boundary for the partitioning, we may encounter DTs even bigger than FoV size horizontally/vertically or in both. Therefore, any slight overlap with such tile incurs large pixel redundancy. In this process, we first define maximum allowable DT size considering the FoV distortion variation according to its vertical position. For example, viewport located towards polar region allows to have a larger DT size as the corresponding FoV on the equirectangular frame spread in a larger region compared to the equator. Hence, considering the # of overlapped tiles with the distorted FoV maps on the equirectangular frame, we define maximum allowable DT size in both vertical (VT_{max}) and horizontal (HT_{max}) directions which is shown in Fig. 7 y-axis.

Note that further reducing VT_{max} and HT_{max} support decreasing redundant data transmission as the DT size becomes small, nonetheless incurs high encoding overhead. To see the impact, we take $\gamma.VT_{max}$ and $\gamma.HT_{max}$ where $\gamma \in [0, 1]$. We set $\gamma = \{0.25, 0.5, 1.0\}$ in our experiments. Decreasing the γ results in smaller tiles. After detecting larger DTs, we start partitioning outwards from the center lines as in example tile in Fig. 7. The reason is that majority of the user VP concentrated around the center of the frame [9, 14]. Therefore, to reduce potential quality changes within the tile, we keep those DTs near center lines non-splitted as much as possible.

Finally, quality allocated tile scheme an be achieved as in Fig. 2-l considering the multiple properties of DTs such as pixel intensity, size and location of the tile. In VAD360 we do not implement proper bit-rate allocation scheme and keep it as a future work. Further interactions with bit-rate allocation is discussed in Section 7.

5 EVALUATION SETUP

5.0.1 Dataset. We develop and validate algorithms in VAD360 leveraging VP traces collected from 30 videos from three different datasets [14, 16, 25]. All videos are in 60s duration with 30fps. VP center is denoted by $< yaw, pitch >$ angle. Each video has 30 users and we take VP traces from randomly selected 20 users to develop tile schemes using VAD360 and the remaining 10 user VPs to validate the performance of VAD360. The selected videos represents 360° video categorization proposed in [7] and are in different genres such as sports, documentary, stage performance etc. generalizing the content of the videos.

5.0.2 Hardware and software setup. We implement VAD360 architecture using Python, which consists of 4500 lines of code, on MacOS–intel Core i9 2.3GHz single core CPU. We use Networkx-2.4 package for implementing the *MNC* algorithm[6]. Videos in HD (1920×1080) and 4K (3840) resolution are encoded using FFmpeg-4.1 in H265 (HEVC) provided by libx265 at default Quantization Parameter (QP)=28 with motion constrained tiling.

5.0.3 Evaluation metrics and comparison benchmarks. We compare VAD360 with 3 fixed tile configurations 4×6 [19], 6×6 [10], and 10×20 [14]. To evaluate VAD360 with viewport-aware streaming, we use two metrics. *i)* % Pixel redundancy before compression: extra pixels in selected DTs (Derived Tiles) but not overlapped with the user FoV using Eq. 1. Higher the value of pixel redundancy, num. of pixel level operations will increase in video coding at the servers and rendering at the client devices. *ii)* Downlink (DL) data volume: Data transmitted by selected tiles in individual user VPs, which impacts the bandwidth saving. Total num. of DTs covering the entire frame when $\gamma = 1$ and 0.5 is near similar to the fixed tile 4×6 and 6×6 cfgs. respectively. We compare VAD360's relative gain to fixed tiling using the above metrics, and denote two cfgs. as **C1**: $\gamma = 1$ to 4×6 and **C2**: $\gamma = 0.5$ to 6×6.

6 RESULTS

We evaluate VAD360 considering the DT (derived tile) distribution on the video frame and individual user VP, % of pixel redundancy before compression, DL data volume and frame processing time.

6.1 Distribution of DTs on the frame and overlap with user VP

We first analyse the no. of DTs generated by VAD360 on each region: $R(FoV_f)$, $R(FoV)$, $R(Buf)$ and $R(OoV)$. Since, the γ value control

[6]For bipartite graph generation for searching maximum independent chords in a given polygon (cf. Section 3.0.3)

the maximum allowable DT size (cf. Section. 4.4), we vary the γ to see the corresponding variation of no. of DT tiles and their size in BTs on each region. Table. 1 reports the averaged results for all the frames in 30 videos. For each γ value, around 31% of DTs on the entire frame covers $R(FoV)$, however, the avg. tile size is 37.3% lower than the DTs in $R(FoV_f)$. This is due to the region expansion of $R(FoV_f)$ during *Partitioning* step (cf. Section 4.3) and the remaining regions on the $VM(\alpha^*, b, f)$ covered by $R(FoV)$ are smaller patches surrounding the $R(FoV_f)$. Second largest tiles are derived in $R(OoV)$ area as the maximum allowable tile size is higher near the upper and bottom region of the ERP frame.

Table 1: DT distribution in different regions: no. of tiles (# T) and avg. tile size in BTs (S) variation by γ

γ	$R(FoV_f)$		$R(FoV)$		$R(Buf)$		$R(OoV)$		**Total**
	# T	S	# T	S	# T	S	# T	S	**# T**
0.25	18	3.2	19	2.8	11	3.0	16	3.6	64
0.50	8	6.8	13	4.0	8	4.0	11	5.4	40
1.00	4	13.4	9	5.6	6	5.6	9	6.4	28

Fig. 8 shows the temporal variation of DT distribution for the entire video duration for $\gamma = 0.5$. Fig. 8(a) and Fig. 8(b) show that DT distribution becomes stable within first 5s. For example, in Fig. 8(a), no. of DTs in $R(OoV)$ start decreasing from 15 to 10, in contrast DTs in $R(FoV_f)$ and $R(FoV)$ starts increasing from 5 to 10 and 10 to 14 respectively. Fig. 8(b) illustrates that, DT size of $R(OoV)$ decreases from 7 to 5 whereas DTs in $R(FoV_f)$ and $R(FoV)$ keep nearly the constant tile size at 6 and 4. These observations conclude that within first 5s, VAD360 generates large $R(FoV_f)$ and high no. of $R(OoV)$ DTs as the user VPs are concentric to a certain area. As the user VP start spreading on the frame, VAD360 generates more $R(FoV)$ and $R(FoV_f)$ DTs because, no. of blobs with high visual attention have increased, reducing the $R(OoV)$.

Fig. 8(c) shows % user VP overlap with DTs from different regions. We see that, more than 50% and 30% of individual user VP overlaps with DTs from $R(FoV_f)$ and $R(FoV)$ enabling content providers to allocate high quality for DTs in the user VPs. Fig. 8(d) shows the proportion of each DT area overlapped with the user VP. Also, starting from 80%, avg. overlapped proportion of DT tiles on $R(FoV_f)$ decrease to 70%, as at the beginning, $R(FoV_f)$ DTs can provide finer boundary to the individual user VP, but slightly fails with VP dispersion. Only 48% of the area of $R(FoV)$ DTs overlap with user VPs. The reason is many $R(FoV)$ DTs cover boundary of the high visual attention areas which results in low overlap with VPs.

Fig. 9(a)shows the *Total Pixel Intensity per Basic Tile* (PI/BT) of the DTs on 4 different regions on the frame, which corresponds to the visual attention level. Overall, $R(FoV_f)$ and $R(FoV)$ attain more than 1000 PI/BT whereas the majority of DTs in both $R(Buf)$ and $R(OoV)$ have ($\leq$ 500) PI/BT showing the effectiveness of VAD360 threshold selection in *Pre-processing* step. Moreover, Fig. ??(b) illustrates the spatial distribution of PI/BT values of $R(FoV_f)$ and $R(FoV)$. Values are normalized separately for the two regions for the clarity of presentation. The pre-defined Th_f (=0.9) in VAD360 is able to derive the majority of $R(FoV_f)$ tiles at the center of the frame, at where the user viewports are concentrated in general [9, 25]. In the meanwhile, DTs in $R(FoV)$ covers the surrounding regions of $R(FoV_f)$ acting as a high quality buffer to $R(FoV_f)$.

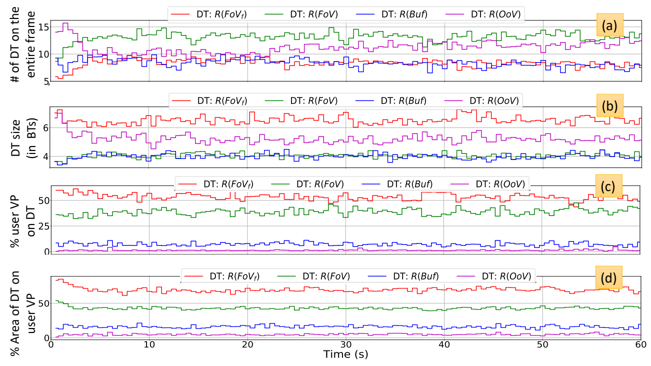


Figure 8: Temporal variation of DT at $\gamma = 0.5$: (a) no. of DT on the entire frame, (b) size of DT, (c) % user VP on DT (d)% area of each DT overlap with user VP

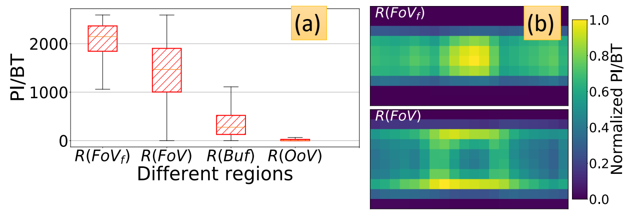
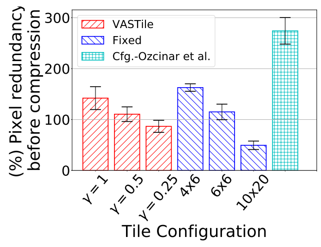
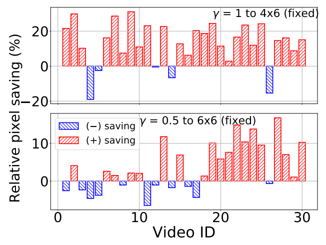


Figure 9: Pixel intensity distribution: (a) Pixel Intensity per Basic Tiles (PI/BT) for the 4 regions, (b) Spatial distribution of PI/BT of $R(FoV_f)$ and $R(FoV)$



(a) Absolute pixel redundancy for the entire video



(b) Relative pixel redundancy saving :VAD360 to fixed cfgs.

Figure 10: Pixel redundancy before compression and relative gain achived by VAD360 compared to fixed cfgs.

Overall, DT distribution on identified 4 regions has unique properties in different aspects such as no. of tiles, size, overlap with user VP and pixel intensity levels. Proper understanding of these properties enables content providers to treat DTs at the servers, adaptively changing their quality levels in order to provide high QoE to the users.

6.2 % Pixel redundancy before compression

We measure *% pixel redundancy before compression* for the entire video using the Eq. 1. Fig. 10a shows the avg. results for all the videos. We see that when $\gamma = 1$ ($\gamma = 0.5$), VAD360 redundant cover is 142 (110)% which is a 20(5)% reduction compared to fixed configuration 4×6 (6×6). Such high redundancy is due to the partial overlap of the user VP by the DTs towards the boundary of the FoV. We further compare the best basic tile configuration provided by Ozcinar *et al.* [18]. Compared to VAD360: $\gamma = 1$, this approach costs additional 137% of pixel redundancy mainly due to the large polar region tiles in their tile scheme.

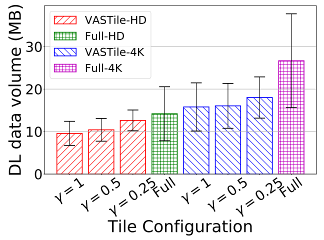
Fig. 10b shows relative saving of redundant pixels before compression in **C1** and **C2** cases for each individual video. We measure an avg. of 12.8% pixel saving and a maximum of 31.1% in C1 ($\gamma = 1$ to 4×6). However the same results for C2 ($\gamma = 0.5$ to 5×8) are 3.7% (avg.) and 16.8% (max). The lower performance in C2 is due to the, reduced size of the fixed cfg. tiles. This is the same reason for negative saving (i.e., more pixel redundancy in VAD360) in certain videos, nevertheless the negative saving is less significant compared to total positive savings. We can not expect the similar patterns in both graphs because, position of the tiles vary when changing the tile cfg. relative to the same user VP, affecting the pixel redundancy.

6.3 DL data volume

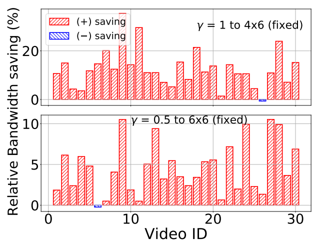
We compare the DL data volume in viewport-aware streaming using VAD360 with streaming the entire video frame for both HD and 4K videos. Fig. 11a shows that, compared to Full frame streaming scenario, which is utilized by commercial content providers such as YouTube and Facebook, VAD360 can save in avg. 32.6 (40.8)% of bandwidth in HD (4K) videos. When decreasing the γ, bandwidth saving reduces to 10.8(32.3)%. The reason is, decrease of γ affects increase in the no. of tiles, and thereby more encoding overhead.

We further measure the bandwidth saving by VAD360 with compared to the existing fix tile cfgs. for each video in Fig. 11b similar to the Fig. 10b. As in the Section. 6.2, **C1** shows more gain, which is 12% in avg. and 35.4% in maximum. The similar measures for **C2** is 4.5% (avg.) and 10.5% (max). Almost all videos show positive bandwidth saving as the compression can boost up the VAD360 performance despite negative relative pixel redundancy (cf. Fig. 10b).

Analysis of pixel redundancy before compression and DL data volume after encoding show that VAD360 outperforms fixed tile cfgs. in general, while achieving significant gains with certain videos. Reduced pixel redundancy benefits in decreasing pixel level operations such as in encoding at the content servers and frame rendering at the client side. Moreover, bandwidth savings compared to full frame streaming enables network providers to relive the high strain on the network by current 360° *video streaming scenario.*



(a) VAD360 with viewport-aware vs Full-frame streaming



(b) VAD360 vs fixed tile cfgs for HD resolution

Figure 11: Comparison of VAD360 with Full frame and Fixed tile cfgs with viewport-aware streaming

6.4 Processing time of VAD360

We measure the end-to-end processing time for VAD360 including *Pre-processing*, *Partitioning* and *Partitioning* steps for the four main regions as shown in Fig. 12(a). In a gist, VAD360 can provide a suitable tile scheme within 0.98(±0.11)s of avg. processing time revealing its scalablity for large scale video datasets. Pre-processing

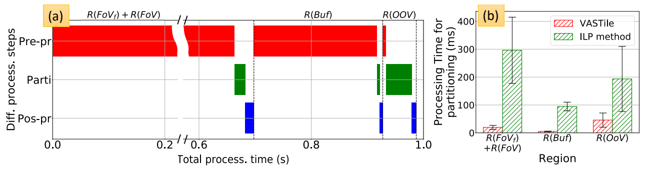


Figure 12: Time complexity: (a) Dissection of VASTile process in time, (b) Comparison with ILP for *Partitioning* step

step in $R(FoV_f)$+$R(FoV)$ and $R(Buf)$ demands higher processing time due to the semi-automated selection of Th_a and Th_{buf}. Comparing the ILP based approach [27] which takes 7−10s to process one frame on single core CPU(3.3GHz), VAD360 can reduce the processing time by 85−90%.

Fig. 12(b) shows a comparison of processing time of *Partitioning* step between VAD360 and modified ILP based method from [27]. Since we consider the minimum number of tiles to cover the video frame, we modify the cost function in ILP method to reduce the no. of tiles whilst keeping *Pre/Post-processing* steps as the same. Overall, *Partitioning* time in VAD360 is 513.2 ms **less** than ILP.

Time for processing a suitable tile scheme by VAD360 is less than 1s showing its potential scalability even for real time streaming. Leveraging an ILP/exhaustive type of approach need additional 0.5s reducing the scalability of the whole process.

7 DISCUSSION AND LIMITATIONS

7.0.1 Server storage. VAD360 does not encode all possible tiles when selecting a suitable tile scheme excluding compression related parameters such as motion vector distribution [27]. Therefore, the solution is sub-optimal reducing server storage optimization opportunities. Incorporating compression awareness to VAD360 should not obstruct the current low processing time. Therefore, we plan to input compression aware parameters to VAD360, utilizing Machine Learning based approach circumventing time consuming steps such as all possible tile scheme compression.

7.0.2 Support for DASH protocol. True benefits of VAD360 can be achieved with a systematic implementation of dynamic bitrate/QP allocation to change the tile quality, in contrast to the constant QP value used at this stage. Compared to fixed tile cfgs, DTs from VAD360 are aware of the viewport distribution on the video frame. Therefore, when adapting to the existing DASH protocols, we can have more control on adjusting the suitable quality affecting parameters (i.e., change the bitrate/QP levels based on the pixel intensity of the tiles on the VMs). We keep DASH implementation of VAD360 and in-detail user QoE analysis as our future work.

8 CONCLUSION AND FUTURE WORKS

In this paper, we proposed VAD360, a viewport aware adaptive 360° video frame partitioning mechanism which derived a suitable tile scheme with variable sized tiles after identifying visually attractive regions on the frame. We leveraged a computational geometric approach to find the minimal non-overlapping cover on these regions and combined the derived tiles on separate regions to cover the entire frame. We evaluated VAD360 in-terms of derived tile distribution both on the video frame and individual user FoVs, saving of pixel redundancy before compression, bandwidth savings and scalability based on the processing time. Our results showed VAD360 outperforms current 360° video streaming scenario with full frame streaming and recently proposed fixed size tiling schemes.

In future work, we aim to investigate on DASH implementation on VAD360 and related QoE aspects. Moreover, we aim to utilize content based saliency maps supporting content providers to encode the frames at a prior stage even without having user viewports.

REFERENCES

[1] 2021 (accessed Feb 26, 2021). Facebook Oculus. https://www.oculus.com.

[2] 2021 (accessed Feb 26, 2021). Facebook360. https://facebook360.fb.com/.

[3] 2021 (accessed Feb 26, 2021). Microsoft Hololens. https://www.microsoft.com/en-us/hololens.

[4] 2021 (accessed Feb 26, 2021). YouTubeVR. https://vr.youtube.com/.

[5] Yanan Bao, Huasen Wu, Tianxiao Zhang, Albara Ah Ramli, and Xin Liu. 2016. Shooting a moving target: Motion-prediction-based transmission for 360-degree videos. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 1161–1170.

[6] Vivek D Bhise. 2011. *Ergonomics in the automotive design process*. CRC Press.

[7] Niklas Carlsson and Derek Eager. 2020. Had You Looked Where I'm Looking? Cross-user Similarities in Viewing Behavior for 360-degree Video and Caching Implications. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. 130–137.

[8] Isha Chaturvedi, Farshid Hassani Bijarbooneh, Tristan Braud, and Pan Hui. 2019. Peripheral vision: a new killer app for smart glasses. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*. 625–636.

[9] Xavier Corbillon, Francesca De Simone, and Gwendal Simon. 2017. 360-degree video head movement dataset. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. 199–204.

[10] Jian He, Mubashir Adnan Qureshi, Lili Qiu, Jin Li, Feng Li, and Lei Han. 2018. Rubiks: Practical 360-degree streaming for smartphones. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. 482–494.

[11] Jeroen Van der Hooft, Maria Torres Vega, Stefano Petrangeli, Tim Wauters, and Filip De Turck. 2019. Tile-based adaptive streaming for virtual reality video. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 15, 4 (2019), 1–24.

[12] Jisheng Li, Ziyu Wen, Sihan Li, Yikai Zhao, Bichuan Guo, and Jiangtao Wen. 2016. Novel tile segmentation scheme for omnidirectional video. In *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 370–374.

[13] Luyang Liu, Ruiguang Zhong, Wuyang Zhang, Yunxin Liu, Jiansong Zhang, Lintao Zhang, and Marco Gruteser. 2018. 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*. 68–80.

[14] Wen-Chih Lo, Ching-Ling Fan, Jean Lee, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. 2017. 360 video viewing dataset in head-mounted virtual reality. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. 211–216.

[15] Rafael Monroy, Sebastian Lutz, Tejo Chalasani, and Aljosa Smolic. 2018. Salnet360: Saliency maps for omni-directional images with cnn. *Signal Processing: Image Communication* 69 (2018), 26–34.

[16] Afshin Taghavi Nasrabadi, Aliehsan Samiei, Anahita Mahzari, Ryan P McMahan, Ravi Prakash, Mylène CQ Farias, and Marcelo M Carvalho. 2019. A taxonomy and dataset for 360° videos. In *Proceedings of the 10th ACM Multimedia Systems Conference*. 273–278.

[17] Anh Nguyen, Zhisheng Yan, and Klara Nahrstedt. 2018. Your attention is unique: Detecting 360-degree video saliency in head-mounted display for head movement prediction. In *Proceedings of the 26th ACM international conference on Multimedia*. 1190–1198.

[18] Cagri Ozcinar, Julian Cabrera, and Aljosa Smolic. 2019. Visual attention-aware omnidirectional video streaming using optimal tiles for virtual reality. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 1 (2019), 217–230.

[19] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. 2018. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. 99–114.

[20] Paul Schmitt, Francesco Bronzino, Sara Ayoubi, Guilherme Martins, Renata Teixeira, and Nick Feamster. 2019. Inferring streaming video quality from encrypted traffic: Practical models and deployment experience. *arXiv preprint arXiv:1901.05800* (2019).

[21] Shu Shi, Varun Gupta, and Rittwik Jana. 2019. Freedom: Fast recovery enhanced vr delivery over mobile networks. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*. 130–141.

[22] Hans Strasburger, Ingo Rentschler, and Martin Jüttner. 2011. Peripheral vision and pattern recognition: A review. *Journal of vision* 11, 5 (2011), 13–13.

[23] Yule Sun, Ang Lu, and Lu Yu. 2017. Weighted-to-spherically-uniform quality evaluation for omnidirectional video. *IEEE signal processing letters* 24, 9 (2017), 1408–1412.

[24] Ohtuski T. 1982. Minimum Dissection of Rectilinear Regions. *Proceedings 1982 International Symposium on Circuits And Systems (ISCAS)* (1982), 1210–1213.

[25] Chenglei Wu, Zhihao Tan, Zhi Wang, and Shiqiang Yang. 2017. A dataset for exploring user behaviors in VR spherical video streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. 193–198.

[26] San-Yuan Wu and Sartaj Sahni. 1994. Fast algorithms to partition simple rectilinear polygons. *VLSI Design* 1, 3 (1994), 193–215.

[27] Mengbai Xiao, Chao Zhou, Yao Liu, and Songqing Chen. 2017. Optile: Toward optimal tiling in 360-degree video streaming. In *Proceedings of the 25th ACM international conference on Multimedia*. 708–716.

[28] Lan Xie, Zhimin Xu, Yixuan Ban, Xinggong Zhang, and Zongming Guo. 2017. 360probdash: Improving qoe of 360 video streaming using tile-based http adaptive streaming. In *Proceedings of the 25th ACM international conference on Multimedia*. 315–323.

[29] Xiufeng Xie and Xinyu Zhang. 2017. Poi360: Panoramic mobile video telephony over lte cellular networks. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. 336–349.

[30] Matt Yu, Haricharan Lakshman, and Bernd Girod. 2015. Content adaptive representations of omnidirectional videos for cinematic virtual reality. In *Proceedings of the 3rd International Workshop on Immersive Media Experiences*. 1–6.

[31] Chao Zhou, Mengbai Xiao, and Yao Liu. 2018. Clustile: Toward minimizing bandwidth in 360-degree video streaming. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 962–970.