A DASH Video Streaming System for Immersive Contents

Giuseppe Ribezzo, Giuseppe Samela, Vittorio Palmisano, Luca De Cicco, Saverio Mascolo
Politecnico di Bari
Bari, Italy
{name.surname}@poliba.it

ABSTRACT

Virtual Reality/Augmented Reality applications require streaming 360° videos to implement new services in a diverse set of fields such as entertainment, art, e-health, e-learning, and smart factories. Providing a high Quality of Experience when streaming 360° videos is particularly challenging due to the very high required network bandwidth. In this paper, we showcase a proof-of-concept implementation of a complete DASH-compliant delivery system for 360° videos that: 1) allows reducing the required bitrate, 2) is independent of the employed encoder, 3) leverages technologies that are already available in the vast majority of mobile platforms and devices. The demo platform allows the user to directly experiment with various parameters, such as the duration of segments, the compression scheme, and the adaptive streaming algorithm parameters.

CCS CONCEPTS

• **Networks** → **Network experimentation**; • **Computing methodologies** → *Virtual reality*;

KEYWORDS

360-degree video; Adaptive video streaming; DASH

ACM Reference Format:
Giuseppe Ribezzo, Giuseppe Samela, Vittorio Palmisano, Luca De Cicco, Saverio Mascolo. 2018. A DASH Video Streaming System for Immersive Contents. In *MMSys'18: 9th ACM Multimedia Systems Conference, June 12–15, 2018, Amsterdam, Netherlands.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3204949.3208107

1 INTRODUCTION

Virtual Reality/Augmented Reality applications are gaining momentum due to the availability of cheap Head Mounted Displays (HMD). In this ecosystem, streaming of 360° videos is a key technology enabling new applications such as immersive cinema, social-media, health-care, and education. Today, video streaming platforms such as Facebook and YouTube are starting to offer 360° videos on their platforms. The challenges that these new services have to face are numerous, among which we cite: 1) standardization of new video formats, 2) design of control algorithms for bitrate selection, 3) design of new compression techniques suitable for 360° videos. In particular, to quantify the impact of the last issue, in [5] authors show that streaming a 360° video requires a network bandwidth of ~400 Mbps to deliver a video quality similar to that of a full HD resolution 2D video.

In an effort to reduce the required bitrate, several approaches have been proposed in the recently published literature. The common feature of these techniques is that only a portion of the video, the one in the current Field of View (FoV) of the user – named the Region of Interest (RoI), is downloaded by the client. In this spirit, the *slicing* technique divides the video into a number of portions which are encoded and stored separately in different bitstreams [9]. The issue of this approach is that a RoI may span multiple slices each one requiring a separate decoding process at the client, making this solution not viable for mobile users. Another issue is that the client has to download in parallel the slices composing the RoI, which makes the adaptive streaming algorithm more complex. A promising approach overcoming the above-mentioned issues is *tiling*, a concept recently introduced in HEVC [8] and proposed for DASH video delivery systems in [1]. The video is spatially divided into a number of *tiles* which are encoded independently and possibly stored into a single bitstream. This way, the client selects a subset of tiles and a single process is able to decode the compressed bitstream. The drawback of this approach is that the representations can only vary the bitrate, but the size of each tile must remain constant, i.e. the tile grid cannot change [1]. Another issue is that tiling efficiency, in terms of required bitrate to obtain the same video quality, is inversely proportional to the number of tiles [7]. Finally, in the case of sudden changes of the viewpoint, new tiles should be downloaded and rebuffering events might occur in the case segments are not downloaded fast enough. From a technological standpoint, the proposed approaches require encoders supporting tiling that are still in the experimental stage[1] and hardware decoding of these new formats is not yet available and deployed in mobile devices. We believe that the issues described above might hinder the penetration of this technology.

In this paper, we present a DASH delivery system for 360° videos that: 1) allows reducing the required bitrate, 2) is encoder-agnostic, 3) leverages technologies that are already available in the vast majority of mobile platforms and devices. We showcase the proposed system through a proof-of-concept implementation which employs a modified version of the Shaka player. The proposed approach is described in Section 2, whereas technical aspects are provided in Section 3.

2 THE PROPOSED SYSTEM

Figure 1 shows the overall architecture of the proposed delivery system which is composed of: 1) a video content generation system;

MMSys'18, June 12–15, 2018, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5192-8/18/06.
https://doi.org/10.1145/3204949.3208107

[1] https://github.com/gpac/gpac/wiki/Tiled-Streaming employs the Kvazaar encoder, an open source implementation of HEVC implementing tiling.

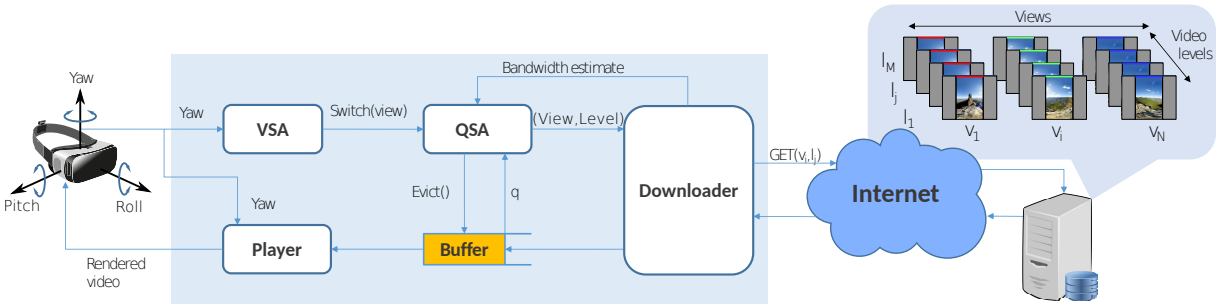

Figure 1: The proposed delivery system architecture

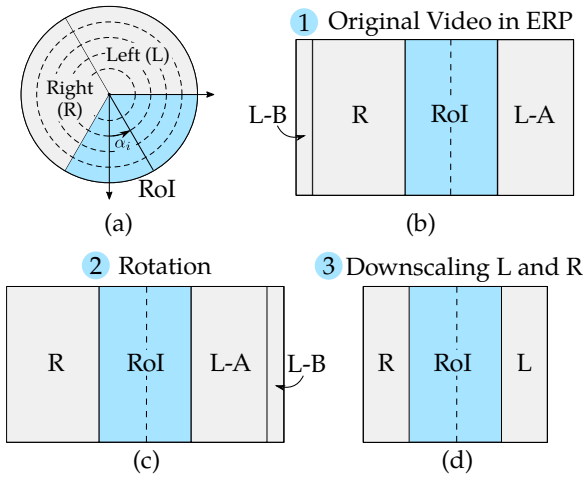

Figure 2: Approach to generate the i-th RoI representation

2) a player running at the client which manages the control logic and the rendering of the received video; 3) an HTTP server providing the video content.

Video content generation. We start by describing the *video content generation* component. In this paper, we consider the original uncompressed scene to be produced in Equirectangular Projection (ERP) format. Notice that this is not a limitation since any other format is in principle supported by using format adapters filters.[2] The original video is manipulated in order to create N different *views* v_i, one for each considered RoI, constituting the *views set* $\mathcal{V} = \{v_1, \ldots, v_N\}$. Saliency maps could be used to tailor the selection of the number and position of the RoIs [2, 6]. The way RoIs are identified is not the focus of this paper. In particular, we assume each RoI to be a *spherical lune* with a dihedral angle equal to 120° (i.e., the FoV angular width) centered at a particular yaw angle α_i as shown in Figure 2(a). The regions outside the RoI, namely the ones at its left (L) and its right (R), are divided into two spherical lunes of equal dihedral angle. Each spherical lune maps to a particular vertical strip of the ERP video (Figure 2(b)). The ERP video is manipulated in such a way that the RoI is always placed in the center of the frame as Figure 2(c) shows. The idea is to downscale the portions of the video outside the RoI, which are less likely to be in the user's FoV, to reduce the required encoding bitrate (Figure 2(d)).

We employ this strategy due to the following reasons. First of all, we use spherical lunes as RoI because more complex strategies (such as ones employing spherical sectors [3]) may introduce inefficiencies into the intra-frame operations, leading to higher bitrate requirements [1, 10]. Moreover, we choose to maintain the RoI at the center of the frame– applying a rotation before downscaling – to better exploit the motion compensation algorithm by keeping the continuity between the scaled and non-scaled areas [1, 10]. Finally, we have chosen to use the *downscaling* operation – instead of *HEVC tiling* – due to the fact that this technique is 1) a *technology-ready* solution, independent of the employed codec, 2) can be efficiently handled by hardware decoders at the client-side, 3) can use well-established and mature algorithms (interpolation, filtering, etc.) to improve the resulting video quality.

Now each of the N views $v_i \in \mathcal{V}$ is encoded into M video representations at different bitrates l_j (and resolutions) constituting the video level set $\mathcal{L} = \{l_1, \ldots, l_M\}$. At the end of this procedure, we obtain the set of representations $\mathcal{R} = \mathcal{V} \times \mathcal{L}$ composed of $N \cdot M$ files that are stored at the server and indexed through an MPD manifest. Technical details on the DASH manifest are provided in the Section 3.

The client. We now describe the client-side part of the proposed system. The client implements both the *control logic* needed to dynamically select which video segment to download and *video decoding and rendering* functionalities. The control logic is composed of two cooperating components: 1) a *view selection algorithm* (VSA) which dynamically chooses the most suitable view representation $v(t) \in \mathcal{V}$ to be downloaded based on measurements provided by the HMD accelerometer; 2) a *quality selection algorithm* (QSA) that dynamically selects the video level $l(t) \in \mathcal{L}$ in order to avoid playback interruptions due to rebuffering while maximizing network channel utilization.

Let us focus begin by describing the way the view is selected. The VSA strives to select the best view representation depending on the current user's head position. In particular, let us assume that the user is currently watching the view $v_1 \in \mathcal{V}$ corresponding to a RoI centered at yaw angle α_1 (see Section 2) and the user turns his head to a yaw angle α. In order to improve visual quality, the algorithm triggers a switch to a view representation $v_i \in \mathcal{V}$ whose

[2] https://trac.ffmpeg.org/wiki/RemapFilter

RoI is centered at a yaw angle α_i that has the minimum absolute distance from α.

The QSA acts similarly to classic DASH adaptive video streaming algorithms. In fact, our approach only needs one playout buffer since the QSA does not have the necessity of downloading in parallel segments of different views (which would need separate buffers to be handled). This is a major advantage compared to other strategies requiring controlling multiple playout buffers since the proposed design significantly simplifies the control logic. The only difference *wrt* classical DASH adaptive video streaming algorithm is the way the QSA reacts to a view switch required by the VSA. Consider the case in which a switch from view v_A to view v_B is triggered by the VSA. At this instant, the playout buffer would store a certain number of segments of the view v_A. Playing such segments would delay the positive effect on visual quality of the triggered view switch. To improve the video quality as soon as possible, it is required to drain from the playout buffer queue an appropriate number of seconds belonging to the segments of the v_A view. Thus, the only additional functionality that QSA is required implement is an algorithm that computes the proper number of seconds (of the v_A view) to be evicted from the tail of the buffer. From this point on, the view switch is completed and the downloader will start downloading segments of the view v_B at the video level $l(t)$ computed according to the control logic implemented by the QSA.

3 DEMO SETUP

In this section, we describe a proof-of-concept that implements the 360° video delivery system presented in Section 2.

3.1 Content generation

The proposed content generation mechanism described in the previous section has been implemented as a filter chain using FFMPEG[3], in such a way that the solution is suitable both for *on-demand* and *real-time* streaming. We have employed 10 benchmark videos at 4K resolution (i.e., 3840×2048) paying attention at selecting contents to obtain a video catalog sufficiently representative of different video categories and features. Each of the three vertical strips in which the video is divided (left, RoI, right) has a resolution equal to 1280×2048. As described in Section 2, for each view $v_i \in \mathcal{V}$ the regions outside the RoI are downscaled in order to reduce the encoding bitrate. We define the *downscale factor* d as the ratio between the width of the downscaled video and the original video width w, i.e. $d = (2w_{\mathrm{d}} + w_{\mathrm{RoI}})/w$, where w_d is the width of the downscaled regions outside the RoI and w_{RoI} is the width of the RoI. In our demonstration we are considering videos with an original width $w = 3840$px, which means that the width of the RoI is equal to $w_{\mathrm{RoI}} = w/3 = 1280$px. We let the downscaled width w_d to vary in the set $\{240\text{px}, 480\text{px}, 720\text{px}, 1080\text{px}\}$.

Table 1 reports the average bitrate reduction (in percentage), along with the 95% confidence intervals, measured for each of the considered downscale factors. The downscaled videos have been encoded using the FFMPEG H.264 encoder with a CRF (Constant Rate Factor) value equal to 20. [4]

[3]https://ffmpeg.org/ffmpeg-filters.html
[4]https://trac.ffmpeg.org/wiki/Encode/H.264

Table 1: Average bitrate reduction (with 95% confidence interval reported in the parentheses) for the considered downscaled resolutions in the case of CRF=20.

Downscale factor d (%)	Downscaled resolution (px)	Average Bitrate reduction (%)
54.17	240 px	51.3 (48.0-54.7)
41.67	480 px	37.49 (34.1-40.8)
29.17	720 px	25.44 (21.9-28.9)
10.42	1080 px	7.90 (3.0-12.7)

```
<AdaptationSet ... maxWidth="3840" maxHeight="2160" ... >
  <Viewpoint schemeIdUri="urn:mpeg:dash:viewpoint:2011"
    value="camera_id=0,adaptive_pano,rotations=0:120:-120,
         viewpoint_id=0, width=1760,side_width=240"/>
    <Representation id="0/dash/0/" ... width="1280" ... />
    <Representation id="0/dash/1/" ... width="1920" ... />
    <Representation id="0/dash/2/" ... width="3840" ... />
  [...]
<AdaptationSet ...  maxWidth="3840" maxHeight="2160" ... >
 <Viewpoint schemeIdUri="urn:mpeg:dash:viewpoint:2011"
   value="camera_id=0,adaptive_pano,rotations=0:120:-120,
          viewpoint_id=1, width=1760,side_width=240"/>
   [...]
```

Figure 3: Excerpt of the DASH manifest

The results show that the proposed approach is promising and provides a percentage bitrate reduction scaling almost linearly with the percentage downscale factor d. Notice also that confidence intervals are quite tight, indicating that the proposed scheme is not content-sensitive.

3.2 DASH Manifest

Our implementation is MPEG-DASH compliant and leverages the `Viewpoint` elements defined by the standard, which are intended for offering multiple camera viewpoints belonging to the same scene (e.g. within Sports events, Concerts, etc). We exploit these descriptor elements to include all the information needed for the client to properly process our video representations. Figure 3 shows an excerpt of the extended DASH manifest file as an example.

The `value` attribute of the `Viewpoint` element is a list of the following comma separated parameters: 1) `camera_id`, identifies a video source (e.g. viewpoints of the same content will have the same `camera_id`); 2) `video_type`, to distinguish scaled panoramic videos (`adaptive_pano`) from non-scaled ones (`panoramic`) and simple 2D videos (`regular`); 3) `rotations`, an ordered list of rotation angles in degrees of all the viewpoints; 4) `viewpoint_id`, the index of the viewpoint in the aforementioned rotations list; 5) `width`, the width of the rescaled video; 6) `side_width`, the width of the rescaled part of the video.

3.3 The Player

The player has been developed by exclusively using standard web technologies and open-source libraries to make sure that it can be run on most modern browsers. The streaming session is handled

by a modified version of the Shaka player[5], an open-source DASH video streaming player written in JavaScript. The player has been modified to support two additional features: 1) parsing of the *mpd* file to extract information regarding the *Viewpoint* element of the manifest; 2) the possibility to evict a certain duration of the video stored in the playout buffer as described in Section 2 when a view change is triggered by the VSA.

Regarding the first feature, the Shaka library only provides support for basic DASH features which do not include the support of viewpoints. For this reason, we expanded the manifest parser of Shaka to extract the viewpoint elements as well. Regarding the second feature, we have modified the Shaka player allowing to programmatically retain a given amount of the playout buffer (the *safety margin*) in order to avoid rebuffering events, and drain the remaining part to speed-up the visual quality improvement due to a view switch.

Finally, a module called *abr-manager* has been written from scratch to handle all the decisions taken by the video streaming control algorithm. This module implements both the logic of the QSA and the VSA. In particular, we employ ELASTIC [4] as the QSA aiming at adapting the video level $l(t)$ to changing network conditions. Regarding the VSA, the *abr-manager* implements automatic switching between viewpoints according to the user head position and movements, and can also switch to entirely different cameras – f.i., a different scene – if the user prefers.

3.4 Video Rendering

The rendering of the received ERP video in VR, as well as the stereoscopic effect needed to support VR headsets, have been implemented using WebVR and the open source library THREE.js.[6] In 3D graphics, a 3D object is composed of two parts: the mesh, modelling the physical properties of the object with vertices, edges, and faces; the texture, which is a composition of one or more images that can provide to the object the perception of realism. The rendering phase of the 3D object includes a mapping of the vertices of the mesh to specific points of the texture. Since our approach downscales the regions outside the RoI, we exploit this process to rescale them back to the original resolution. With this purpose, we leverage the original rendering pipeline to introduce in the Vertex Shader a modified mapping function that properly associates the vertices of the mesh to the differently scaled strips of the video frame. It is important to stress that rescaling the regions outside the RoI does not incur in an added computational cost compared to the usual rendering process involved when remapping a standard ERP video. In fact, the performed operation is equivalent to rendering the ERP video at the original resolution, but using a different mapping.

3.5 Experimental Scenarios

The demo setup allows the user to experiment with several system parameters and network conditions and perceive the impact on the overall experience. The user will wear a VR headset equipped with a mobile phone which will run the client-logic in the mobile version of the Google Chrome web browser.

[5]https://github.com/google/shaka-player
[6]https://threejs.org

In particular, user will be confronted with several scenarios varying: 1) the video segments length, in order to assess the impact of delayed actions (representation switches can only occur at the segment granularity); 2) video contents, ranging from more static to highly dynamic scenes with multiple RoIs; 3) network conditions in terms of changing available bandwidths throughout the experiment set with network emulation tools; 4) downscale factors d (as reported in Table 1) to assess the visual quality in downscaled regions outside the RoI. In reason of the large parameter space, we will provide the user a selected number of interesting scenarios.

4 CONCLUSIONS

In this paper, we describe a practical implementation of a 360° delivery system based on technologies that are available on the vast majority of user devices. The delivery system is showcased through a proof-of-concept implementation which allows experimenting with various trade-offs involved when tuning system parameters. In particular, duration of segments, the compression parameters, and adaptive streaming strategies are considered. Their impact can be directly assessed by users employing the proof-of-concept system which makes use of a network environment emulating several bottleneck capacities and background traffic scenarios.

ACKNOWLEDGEMENT

This work has been partially supported by the Italian Ministry of Economic Development (MISE) through the CLIPS project (no. F/050136/01/X32). This work has also received funding from the Apulia Region (Italy) through the "Future in Research" programme, project no. ACYBEH5. Any opinions, findings, conclusions or recommendations expressed in this work are the authors' and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] C. Concolato, J. Le Feuvre, F. Denoual, E. Nassor, N. Ouedraogo, and J. Taquet. 2017. Adaptive Streaming of HEVC Tiled Videos using MPEG-DASH. *IEEE Trans. on Circuits and Systems for Video Technology* PP, 99 (2017), 1–1. https://doi.org/10.1109/TCSVT.2017.2688491
[2] X. Corbillon, F. De Simone, and G. Simon. 2017. 360-degree video head movement dataset. In *Proc. of ACM Multimedia Systems Conference*. 199–204.
[3] X. Corbillon, G. Simon, A. Devlic, and J. Chakareski. 2017. Viewport-adaptive navigable 360-degree video delivery. In *2017 IEEE International Conference on Communications (ICC)*. 1–7. https://doi.org/10.1109/ICC.2017.7996611
[4] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. 2013. ELASTIC: A Client-Side Controller for Dynamic Adaptive Streaming over HTTP (DASH). In *Packet Video Workshop (PV), 2013 20th International*.
[5] S. Mangiante, G. Klas, A. Navon, Z. GuanHua, J. Ran, and M. D. Silva. 2017. Vr is on the edge: How to deliver 360 videos in mobile networks. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*. ACM, 30–35.
[6] Y. Rai, J. Gutiérrez, and P. Le Callet. 2017. A dataset of head and eye movements for 360 degree images. In *Proc. of ACM Multimedia Systems Conference*. 205–210.
[7] Y. Sanchez, R. Skupin, and T. Schierl. 2015. Compressed domain video processing for tile based panoramic streaming using HEVC. In *2015 IEEE International Conference on Image Processing (ICIP)*. 2244–2248. https://doi.org/10.1109/ICIP.2015.7351200
[8] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand. 2012. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Trans. on Circuits and Systems for Video Technology* 22, 12 (Dec 2012), 1649–1668. https://doi.org/10.1109/TCSVT.2012.2221191
[9] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. 2003. Overview of the H.264/AVC video coding standard. *IEEE Trans. on Circuits and Systems for Video Technology* 13, 7 (July 2003), 560–576. https://doi.org/10.1109/TCSVT.2003.815165
[10] R. G. Youvalari, A. Aminlou, M. M. Hannuksela, and M. Gabbouj. 2016. Efficient Coding of 360-Degree Pseudo-Cylindrical Panoramic Video for Virtual Reality Applications. In *2016 IEEE International Symposium on Multimedia (ISM)*. 525–528. https://doi.org/10.1109/ISM.2016.0115