A Practical and Scalable Method for Streaming Omni-Directional Video to Web Users

Peter Quax, Jori Liesenborgs, Panagiotis Issaris and Wim Lamotte
Hasselt University - tUf - iMinds
Expertise Center for Digital Media
Wetenschapspark 2, 3590 Diepenbeek, Belgium
peter.quax@uhasselt.be

Johan Claes
Androme NV
Wetenschapspark 4
3590 Diepenbeek, Belgium

ABSTRACT
With the advent of low-cost capturing hardware, omnidirectional or panoramic video is likely to become a more popular delivery format for video content in the near future. Besides the inherent advantage of being able to capture anything within a scene without intervention from a director or camera operator, the format also increases the interactivity for the end-user and may result in a more immersive experience. To stream this kind of content to a wide range of end-users, a web-based delivery platform is clearly preferred. Unfortunately, a lack of uniform codec and streaming container format support hinders practical deployment, especially for live content distribution. In this paper, a solution is proposed that overcomes these issues for most browsers currently in use by desktop computers. It consists of a back-end module that encodes and encapsulates content into a WebM compliant streamable format in real-time, which is subsequently decoded by the built-in browser decoders and displayed through an HTML5 video element - either through a planar projection in 2D or through a 3D spherical texture transformation in WebGL. The proposed solution has been deployed on an existing CDN and tested for scalability in the context of a day-long event with omni-directional webcasts.

Categories and Subject Descriptors
H.4.3 [Information Systems Applications]: Communications Applications

Keywords
omni-directional video; streaming video; scalability

INTRODUCTION AND RELATED WORK

Omnidirectional video (ODV) is a relatively new paradigm that enables video to be captured in 360 degrees. One can compare this technology to Google’s Streetview initiative – however using video rather than still images. The applications of this technology are wide-ranged, from surveillance over capturing sports events, concerts to guided tours [11, 2]. Within the iMind sICON XTV project, possible applications of this technology were explored in the context of a.o. second screen applications, where the ODV content supports a more traditional TV broadcast setup [7]. Adjacent projects have focused on the same topic, but use other (specialized) distribution methods [8, 1]. ODV technology itself however is useful in a stand-alone solution as well, and is currently studied as such in the iMinds ICON AIVIE project.

Current solutions for web casting are typically hindered by the need for multiple camera’s to be deployed, to be able to record both the contents of a projection screen, as well as multiple speakers located throughout a scene. Similarly, in a broadcast conferencing setup, tracking multiple speakers around a table requires either manual intervention or a movable camera setup. By recording the entire scene from a single location, but in 360 degrees, the system is inherently capable of recording everything that happens.

Many peripherals have come on the market the past few years to enhance existing capturing equipment with the ability to record 360 degree panoramas. This ranges from optical extensions for smartphone devices [4] to high-end dedicated multi-camera capturing solutions [6]. One can therefore state that the capturing part of the end-to-end chain is covered quite well. The same is currently not true for the delivery of the content to the end-user. Typically, custom solutions are proposed (in the form of specialized software packages or dedicated plug-ins) that present compatibility issues with existing networks and applications.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Immersive Media Experiences 2013 Barcelona, Spain
ACM 978-1-4503-2402-1/13/01 ...$15.00.
http://dx.doi.org/10.1145/2512142.2512149
Figure 1: High-level overview of the end-to-end delivery chain

2. THE PROBLEM OF BROWSER SUPPORT

The HTML5 standard is currently in the Candidate Recommendation phase (as announced by W3C [10]). This new version of web standards enables the integration of multimedia elements within web pages without requiring external plugins or software such as Adobe Flash. Relevant new elements to the technology presented here are <audio>, <video> and <canvas>. What W3C did not do to do was to standardize the codec and container formats to be used by browser vendors; the video element is a generic container in itself that merely links to an URL where video frames can be obtained. The practical issue is that many browser vendors have opted to make their browser compatible only with a limited subset of today’s state of the art codecs and containers. Because of many reasons, Google with its Chrome browser for example prefers to use WebM technology [9] - which consists of the VPS video codec, Vorbis audio and a custom container. Apple’s Safari browser includes support for its HTTP Live Streaming [5] format, which incorporates H.264 as video codec, an MPEG-2 container and a manifest-based chunk delivery algorithm over HTTP. More generic solutions for delivery over HTTP have also been developed, such as MPEG-DASH [3], but support for these in browsers is currently non-existing. Although there is some cross-over compatibility (e.g. Chrome enables playback of H.264 content), a universal solution that is applicable to all browsers and usage scenarios (live and pre-recorded) is not available.

Within these practical constraints, the greatest common divisor regarding codecs is the use of WebM, as decoding is currently supported in both WebKit and Gecko based browsers (i.e. most importantly Chrome and Firefox, which combined account for over 60% of desktop browser marketshare). Other browsers do not ship with WebM support by default (e.g. IE 9), but the codec is likely to be installed by external software as part of an installation package. For the application scenario envisaged (being live webcasts) however, WebM unfortunately lacks support for delivery of this type of content in its container format (which is based on the Matroska container).

3. PROPOSED SOLUTION

3.1 Back-end

The architectural design is represented in figure 1. The first component in the (content generating) back-end is the capturing infrastructure. Although the solution presented here is compatible with a wide array of capturing hardware (as will be shown later), for practical purposes a high-end dedicated camera setup was used for testing, along with a custom developed software package that is capable of stitching together multiple HD video streams into a single output frame. This frame is an equirectangular projection of the 360 degree content; a format that is suitable for both planar visualization, as well as spherical or cylindrical texturing with minimal deformation. The output of this capturing system is subsequently fed into the webcast encoder. To minimize the latency between the stitching hardware and the encoder, an HDMI link is utilized. This approach avoids additional compression and decompression of streams and delivers a video of sufficient quality for webcasting purposes. The encoder is equipped with an HDMI capturing card that is capable of retrieving Full-HD input frames in real-time (i.e. a Blackmagic Intensity Pro PCI express card).

Figure 1 also indicates that the system is capable of delivering both high- and low-quality versions of the output stream, which can be fed individually to multiple destinations through a variety of connections (e.g. for transmitting the stitched video in original non-compressed 4K or higher resolutions, 10GE links can be utilized). Depending on the use case, the system is also flexible enough to perform processing tasks remotely (i.e. non-colocated with the capturing infrastructure). In this latter case, the high-quality video is typically sent to a separate infrastructure for encoding/processing/delivery, while a lightly compressed stream can be sent to the WebM encoder, transcoded in real-time and processed (chunk generation) for delivery to the CDN.

Once the frames are available in the encoder, they are processed to generate chunks of video data of a specific duration (i.e. about 5 seconds of video within a chunk). This approach is similar to Apple HLS or MPEG-DASH, in which a manifest is created that represents the individual chunks that make up the entire video stream. First of all, a generic WebM (container) header is generated and written to a file. Next, video frames are encoded using the VPS codec, available through the FFmpeg framework. Once a chunk is complete, it is written to a separate file, always taking care that a chunk starts with a keyframe (the codec is instructed to generate video frames with an appropriate GOP size to accommodate this).

The next step consists of uploading the files to the CDN as they become available (starting with the header file). To trigger the upload, the linux Inotify system is utilized, which couples user-defined actions to a specific file system operation (i.e. the creation of a new chunk). In theory, any upload mechanism can be utilized between the encoder and the CDN such as FTP or SSH, however it is typically more efficient to use specific extensions provided by the CDN. In this case, an HTTP interface was available that allowed for uploading the content using a custom python script and CURL bindings. It is essential to note here that - as far as the CDN is concerned - only files need to be synchronized between the nodes and not real-time streams. Such an approach is different to e.g. Flash-based solutions that require specific support at the video server end (e.g. compatibility with the proprietary Adobe Media Encoder & Server technology). Given the fact that, typically, CDN systems are highly optimized to distribute content stored in individual files, scalability can be ensured.

Once a client connects to the CDN node, it requests a specific URL through the HTML5 video element (more on this in the next section). The CDN node runs a custom CGI script that performs the following actions:

1. The header information (which is generic for all chunks) is sent to the client first
2. The five most recent chunks are transmitted in-order (oldest to newest). In case there are fewer than five chunks available (but at least one complete), the complete ones are sent immediately without additional wait times.
3. Additional chunks are transmitted to the client as they become available (when uploaded to the CDN by the encoder).

It is essential to note that the web browser only ‘sees’ a continuous stream of video data and is not aware of the chunk boundaries (as demonstrated in figure 2, with N representing the most recent complete chunk available at the time of the initial request). This
approach is required as the WebM container has no provisioning for live video transmissions. By adopting this solution, the browser is tricked into playing data that is being generated on-the-fly, rather than pre-recorded. By conforming to the WebM format, the browser can decode the frames using built-in functionality and make them available to the web developer or user through the interface of the `<video>` element. For typical usage scenarios with small viewport sizes, a bandwidth of 1.5mbps allows for sufficient user freedom in terms of visual quality and interactivity (i.e. zooming in and out).

3.2 Front-end

```html
<html>
  <head>
    <script src="three.js"></script>
    <script src="odvviewer.js"></script>
  </head>
  <body onload="init()">
    <video id="video1" autoplay hidden="hidden" crossorigin="anonymous">
      <source src="http://edm.fast-tool01.rambla.be/webstream" type="video/webm">
      Your browser does not support HTML5 video.
    </video>
    <div id="container1"></div>
    <div id="container2"></div>
    <script>
      var viewer1 = null;
      var viewer2 = null;
      function init() {
        viewer1 = new ODV.Viewer('video1', 'container1', 66, 320, 240, false);
        viewer2 = new ODV.Viewer('video1', 'container2', 66, 320, 240, true); // will only use WebGL if possible
      }
    </script>
  </body>
</html>
```

Figure 3: Code sample for including ODV within a web page

As indicated in the previous section, the client receives a continuous stream of data that is being concatenated from individual chunks by the CGI script running on the CDN nodes. However, typically, it does not suffice to simply display this data in a `<video>` element. Because of the need for interactivity (i.e. the user manipulating the viewport during the webcast), the video data needs to be copied to a `<canvas>` element, which enables interaction and displays only the parts of the video that are relevant for the user (either using a 2D projection or WebGL).

Javascript functionality is provided to dynamically insert a `<canvas>` element in a web page inside a pre-defined `<div>`. A hidden `<video>` element is included in the web page which links to the WebM stream on the CDN node. Both are handed as parameters to a Javascript function that takes care of extracting the frames from the `<video>` element and including them in the dynamically inserted `<canvas>` object. An outline of this procedure is shown in figure 3.

As the browser is only aware of a single file that is being delivered, advanced buffer management based on a dynamically generated manifest for live transmissions (like in HLS or MPEG-DASH scenarios) is not needed. Also, buffering delay at server- and client-side is minimized compared to these technologies. In HLS, a live manifest is typically required to contain at least three valid segments (=chunks). At the same time, the client needs to buffer at least these three segments for playback to start. In case...
of the proposed solution, the server needs to have access to at least one valid chunk before the CGI script can start transmitting valid data to the browser. Although the delay associated with the browser’s built-in buffering mechanism cannot be tweaked, it is typically below the chunk duration (given the typical scenario of about 5 seconds in each chunk) – Chrome starting playback without any noticeable delay. Clearly, the solution adopted here is vulnerable to playback dropouts due to buffer underruns; care needs to be taken to ensure that typical end-user bandwidth capacity is not exceeded. As possible mitigation, multiple quality versions of the same content can easily be provided by the same infrastructure to stay within these limits (see below for a discussion on real-time adaptation).

Another restriction to be aware of concerns the 3D visualization of content using WebGL. In this case, a sphere is constructed (e.g. using an external library such as three.js) and textured on the inside using the obtained video data. The virtual camera is subsequently positioned in the center point of the sphere. However, because of security concerns, many browser developers do not allow for the video content to be served from a different origin to the <canvas> element it is displayed in. This restriction can only be lifted through two means: either include the crossorigin="anonymous" attribute in the <video> element or add the Access-Control-Allow-Origin field in the HTTP header of the video stream.

The Javascript library that takes care of the visualization is also responsible for interaction support. Typical interaction methods for 3D scene navigation (i.e. drag-and-drop to rotate the virtual camera, mouse scroll wheel support for zoom level manipulation) are handled through events that are bound to the HTML <canvas> element. The choice between 2D visualization using the planar projection or the 3D spherical texture projection (using WebGL) is dependent on a number of factors. First, the processing power required for the WebGL is typically higher, due to the additional computation required for texture mapping. Second, not all browsers have native support for (accelerated) WebGL and may impose additional security restrictions. However, the latter can be mitigated by a built-in fallback mechanism that switches to the 2D planar rendering method in case WebGL support is absent. Finally, the content that is being displayed may play a role in the decision. For some content types, the visual distortion created by the 2D rendering can actually be used to an advantage. This projection is capable of zooming out to encompass the entire captured frame size, something that is not possible using the 3D rendering method. Users have reported that it is sometimes easier to navigate through an ODV scene if the system enables such zooming operations (i.e. it helps the user to get his/her bearing within the scene).

Example screenshots are shown in figure 4. The visual distortion is clearly shown in 4(b) and absent in 4(c).

3.3 Run-time adaptation

Just like with Apple HLS and MPEG-DASH, run-time adaptation of quality could also be achieved using the proposed combination of technologies. In this scenario, a server-side CGI script is responsible for listening to client feedback on bandwidth and quality requirements. This information is stored (for each connected client individually) in the back-end for association with a specific stream currently being transmitted to the client. In case there is a change in the requirements of the client, the web browser will send a request to the adaptation CGI script to change the properties of the stream currently being sent towards the client. It should be clear that such an approach requires multiple versions of the content to be available server-side (just like with HLS). A disadvantage in comparison to the mechanism built into HLS is that dynamic resolution switching is not easily achieved, as the browser decoder context is unlikely to be adjustable for other resolutions at run-time. The reason to perform stream switching in the back-end rather than in the browser (by e.g. dynamically replacing the source attribute of the <video> element) is that precise synchronization is very difficult to achieve within the browser environment through standard-compliant mechanisms. While minor disturbances might be acceptable for video sequences, hiccups or synchronization artefacts in the audio stream would severely disturb the user experience when used for inter-user communication purposes.

4. CONCLUSION AND FUTURE WORK

In this paper a scalable distribution method has been proposed that enables users to interact with live omni-directional video streams through a standard web browser. Practical workarounds are discussed that overcome major browser incompatibilities in terms of codec and container incompatibility, which is especially relevant in the context of live transmissions. The solution is an end-to-end architecture that encompasses both the content generating back-end as well as the components required for end-user-facing web browser integration. By adhering to established web standards, compatibility with existing CDNs is easily achieved and scalability of the system can be guaranteed. The system has been validated through practical deployment in an all-day event with live webcasts supporting over 200 unique users on a single CDN node, with most viewing sessions lasting multiple hours.

Acknowledgment

This work is part of the iMinds ICON XTV and AIVIE projects. Both are co-funded by iMinds (Interdisciplinary institute for Technology), a research institute founded by the Flemish Government. The authors would like to thank Rambla for their support in configuring the CDN nodes to enable the proposed solution to be tested in practice.

5. REFERENCES


This work is part of the iMinds ICON XTV and AIVIE projects. Both are co-funded by iMinds (Interdisciplinary institute for Technology), a research institute founded by the Flemish Government. The authors would like to thank Rambla for their support in configuring the CDN nodes to enable the proposed solution to be tested in practice.