Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TÍTOL DEL TFG: Platform for innovative content distribution

TITULACIÓ: Grau en Enginyeria Telemàtica

AUTOR: Einar-Martín Meyerson Uriarte

DIRECTOR: Joan Llobera

SUPERVISOR: David Rincón

DATA: 26 Sept 2016

Titulo: Plataforma per la distribució de contingut innovador

Autor: Einar-Martín Meyerson Uriarte

Director: Joan Llobera

Fecha: 26 de Septiembre 2016

Resumen

Hoy en día los usuarios están empezando a consumir nuevos contenidos de carácter innovador en un entorno doméstico debido a la introducción de dispositivos como los cascos de realidad virtual (RV). El problema que afrontamos en este trabajo es cómo acceder a este contenido de RV.

Este trabajo ha consistido en desarrollar una plataforma de distribución de contenidos innovadores utilizando diferentes técnicas de distribución para un proyecto Europeo del Horizon 2020 denominado ImmersiaTV que va redefinir la cadena de distribución amplia extremo a extremo: producción, distribución y entrega de contenido multiplataforma sincronizado basado en video omnidireccional. En el contexto de ImmersiaTV, nuestra contribución se centra en la creación de la segunda pieza del puzle: la distribución del contenido.

Ahora que la producción de contenido de RV este en auge, este tipo de plataformas empezarán a salir a luz ya que la demanda crecerá y por lo tanto la oferta tendrá que seguir el mismo camino. Con esta plataforma queremos crear un sitio en la red donde usuarios que quieran consumir específicamente contenido de RV puedan hacerlo de manera ágil y sencilla.

En este trabajo nos hemos centrado en contenido basado en videos omnidireccionales (para el proyecto ImmersiaTV) y modelos 3D creados por sensores (para este proyecto en particular), cuya publicación y distribución se realiza a través de una aplicación web.

El contenido basado en vídeo omnidireccional se produce con un plug-in diseñado en ImmersiaTV para la herramienta Premiere Pro de Adobe y el contenido basado en modelos 3D, hechos por sensores, se produce con sensores como Kinect 2 y Structure dentro de este mismo trabajo.

Los contenidos tradicionales como el video omnidireccional y el modelado por ordenador son contenidos que, por una parte, son muy rápidos de producir (video omnidireccional) a expensas de la calidad de la experiencia o muy costos en tiempo de producción (modelado por ordenador). Por otra parte los contenidos basados en sensores son rápidos de producir, con un resultado satisfactorio en términos de la calidad de la experiencia de RV.

Para probar el uso de la plataforma hemos usado un reproductor existente creado dentro del proyecto ImmersiaTV. Sin embargo, para probar los formatos innovadores basados en sensores, hemos desarrollado una aplicación específica. Esta aplicación también permite a los usuarios comparar como cambia la experiencia de RV a través de tres diferentes formatos de contenido como son el video omnidireccional, el modelado con ordenador y el basado en sensores.

Los usuarios también podrán experimentar la diferencia entre ser representados por un avatar modelado por ordenador o por un modelo 3D creado en vivo por la Kinect 2 dentro de la experiencia de RV.

Aunque este trabajo no trata de medir los resultados de la experiencia de RV sino de distribuir el contenido para poder ser consumido, durante la creación de la plataforma hemos podido comprobar la gran diferencia, en términos de calidad de la experiencia, entre los diferentes escenarios.

Title: Platform for innovative content distribution

Author: Einar-Martín Meyerson Uriarte

Director: Joan Llobera

Date: 26 Sept 2016

Overview

Nowadays users are starting to consume new innovative content in a domestic environment due to the introduction of head mounted displays. The problem we face in this work is how to access this virtual reality (VR) content.

This work consists in develop a platform for innovative content distribution using different techniques for the European Horizon 2020 project ImmersiaTV which is redefining the end-to-end broadcast chain: production, distribution and delivery of multiplatform synchronized content based in omnidirectional video Inside ImmersiaTV, this work will focus in the second piece of the puzzle, the creation of the content distribution platform.

Now that the production of VR content is booming, these kind of distribution platforms are beginning to come to light as the demand of VR content is growing, therefore the offer has to follow the same path. What we want to create with this platform is a spot in the internet for users who want to consume VR content have a quick and easy way to access it.

This work has focused on content based on omnidirectional videos (for ImmersiaTV) and sensor-generated mesh (for this particular work), whose publication and distribution is done through a web application.

The omnidirectional video-based content is produced with an Adobe Premiere Pro plug-in developed in ImmersiaTV. The sensor-generated mesh content is produced with sensors like Kinect 2 and Structure in this work.

Traditional content like omnidirectional video and computer-generated imagery are content that on the one hand are very quick to produce (omnidirectional video) at the expenses of the experience quality or very costly in production time (traditional CGI), moreover, sensor-generated mesh content are quick to produce with satisfactory results in terms of the VR experience quality.

In order to test the platform we have used an existing player created in the ImmersiaTV project. However, to test the innovative formats, we have developed a specific application. This application also allows users to compare the difference between the VR experience through three different content formats such as omnidirectional video, traditional CGI and sensor-based.

Users can also experience the difference between being represented by an avatar modelled by computer or a live 3D model created by the Kinect 2 inside the VR environment.

Although this work does not attempt to measure the results of the user's VR experience if not to distribute the content in order to be consumed, during the development of the platform we have seen a big difference, in terms of experience quality, between the different content scenarios.

ÍNDEX

INTRODUCTION

Consumer market-oriented virtual reality (VR) headsets have brought Immersive Virtual Environments to the general public. This has provoked the appearance of new methods to create content.

Against the traditional CGI approach, based on hand-crafting 3D virtual environments, omnidirectional video has recently gained momentum as a possible production technique. However, new methods using sensors are starting to gain strength, with different advantages and drawbacks. The big debate today is production speed versus quality of the experience. Although in terms of the experience the Place illusion (the sense of "being there") and the Plausibility illusion (the sense that what you see is actually occurring) are very important [1], nowadays it is unfeasible to create a VR environment that meets the requirements for a perfect experience without tedious and complex hand crafted content production. This is why content producers are searching for new methods that increase production speed without affecting too much the experience.

However, all this new content, along with the traditional one, is difficult to find for general public consumption. Furthermore, the range of devices in the market, with a great variety in resolutions, has increased massively, forcing media content producers to search for solutions to serve best quality content no matter what characteristics and capabilities the different terminals have.

The goal of this work is to satisfy the need for a platform to publish and distribute innovative content. We focus on delivering streamed experiences based on omnidirectional video and sensor-generated mesh content. We developed a web application for distribution of innovative content in four development stages.

To test this web application, we have used omnidirectional videos readily available –produced within the Horizon 2020 Project ImmersiaTV [2] [3] (in Annex 1 you can find a brief explanation of the project). To test this web application using innovative content, we have developed specific content examples using sensors like Kinect 2 and Structure sensor.

Specifically, our contribution in the ImmersiaTV project has been the development of the connexion between the content producers and consumers. We have developed a content distribution platform which converts to DASH content exported from the Adobe Premiere plug-in in order to be published and viewed through the ImmersiaTV Unity player.

This document is divided into four chapters. The first one describes the state of art in different distribution techniques like: RTSP, DASH and AssetBundle. Then it describes different VR content formats, including some innovative ones that have been produced during the development of this work.

The second chapter lists the different technologies, hardware and software used. Hardware like the omnidirectional cameras, the Head-Mounted Display (HMD) or the sensors used to create the innovative formats and the software used to develop the web application platform.

The third chapter will describe the evolution of the distribution platform through the four different stages of development, describing the purpose, architecture, problems encountered and user experience of each stage.

The fourth chapter will describe a demo created in order to test the distribution platform. Moreover, this demo will test the difference in the user's Place illusion (PI) and Plausibility illusion (Psi) between three different environments: omnidirectional video, traditional CGI and sensor mesh-based.

In the conclusions, we describe the final workflow: users can convert content to MPEG-DASH, publish the converted content and view it through a multi-platform player. We also describe a demo developed in Unity 3D with three different scenes representing different VR content formats, where users can compare the quality of experience in each of the VR scenes.

CHAPTER 1. STATE OF THE ART

1.1. Distribution techniques

The purpose of this work is the creation of a platform for distribution of innovative VR content. All this content must be distributed using different methods, depending on the contents' format. In this work we will take advantage of three distribution techniques: RTSP, MPEG-DASH and AssetBundles.

1.1.1. RTSP

Real Time Streaming Protocol [4] (RTSP) is a standard protocol widely used for efficiently control the streaming of audio and video data, particularly in broadcast production environments. The protocol is used either for multicast and unicast streaming. RTSP acts like a remote control for multimedia servers; through the network clients can *play, pause, record* or perform other actions. The transmission of the streaming data is not done by the RTSP protocol. Most of the RTSP servers use Real-time Transport Protocol (RTP) in conjunction with Real-time Control Protocol (RTCP) for media stream delivery through a combination of TCP and UDP. Distribution over the internet is problematic due to the fact that RTP is generally based on the UDP protocol, although RTP over TCP is also possible, and useful for wireless distribution.

Unlike HTTP which was designed for general web content, RTSP was designed specifically for media streaming. Furthermore RTSP/RTP can conceivably signal and carry video and audio of any compression format. Years ago manufacturers tended to stick to the simplest format with the widest web-browser compatibility, JPEG over HTTP, however, due to the customer demand for more sophisticated and mobile-friendly compression formats, there has been a boom in MPEG-4 and H.264 over RTSP/RTP or RTSP/TCP implementations in network cameras [5].

Despite all these advantages, one drawback RTSP has is the fact it works on a different port to HTTP (554 compared to 80) it is sometimes blocked by proxy server and firewalls.

An example of servers and clients using RTSP are GStreamer, VideoLAN with VLC media player and FFmpeg [6].

1.1.2. DASH

Dynamic Adaptive Streaming over HTTP (MPEG-DASH) [7] is an adaptive bitrate streaming technique that allows high quality media content streaming over the internet delivered from HTTP servers. MPEG-DASH works by breaking the content (e.g. movie or live broadcast event) into a sequence of small file segments, each segment containing a short interval of time. Each segment of time is coded for different qualities of video in parallel in order to have all the intervals of time synchronized between them. The MPEG-DASH client detects the user’s bandwidth capacity and adjusts the quality of the video stream between multiple bitrates and/or resolutions. This has made possible, for content publishers, to generate a single set of files that should be compatible with as many devices as possible, and adapt to the bandwidth variations of common IP networks.

Unlike other distribution protocols with adaptive bit rates like Apple HLS, Microsoft Smooth Streaming or Adobe HDS, MPEG-DASH is an open standard and much more flexible in terms of codecs (audio and video), protection and segment duration.

MPEG-DASH is available natively in Android through ExoPlayer [8], an application level media player for Android, on a variety of smart TVs. MPEG-DASH is gaining strength due to the variety of devices with different resolutions there are nowadays in the market. Furthermore, major video distributors like Netflix and YouTube already support MPEG-DASH [9].

1.1.3. AssetBundles (Unity3D format)

AssetBundles [10] are compressed files you can export from Unity3D which allow on demand streaming and loading of Assets from local or remote location. An Asset is any item (e.g. 3D model, audio file or image) that can be integrated within the Unity3D [11] game engine. With AssetBundles, Assets can be stored remotely and accessed on demand, reducing the initial project size.

Assets are compressed before been remotely uploaded as an AssetBundle. Unity3D supports three compression options [12]:

- LZMA – The Lempel-Ziv-Markov chain algorithm (LZMA) is an algorithm used to perform lossless data compression. It was the first used in the 7z format [13] of the 7-Zip archiver. LZMA uses a dictionary compression algorithm[1] similar to the LZ77 algorithm, whose output is then encoded with a range encoder[2], using a complex model to make a probability prediction of each bit [14].

  Compression in a single LZMA stream of serialized data files needs to be decompressed entirely before use. This compression algorithm gives the smallest possible compressed size. However, it decompresses slowly resulting in high loading times.

- LZ4 – LZ4 is a lossless data compression algorithm from the LZ77 family of byte-oriented compression schemes focused on compression and decompression speed [15]. This method produces a larger compressed file size. However, it does not require downloading the entire file to be

[1] Lossless data compression algorithm which operates by searching for matches between the text to be compressed and a set of strings contained in a data structure called the ‘dictionary’.

[2] It is a lossless compression method defined by G. Nigel N. Martin in 1979 which given a stream of symbols and their probabilities, a range coder produces a space efficient stream of bits to represent these symbols.

decompressed before use. This algorithm, unlike LZMA, divides the Asset into compression blocks reducing in some way the decompression time.

- Uncompressed – This method produces the biggest file size. However this format is the fastest to access once downloaded.

Once the AssetBundle is downloaded for the first time, Assets can be cached in order to eliminate delays associated with download.

There are many use-cases for AssetBundles [16]:

- New content can be dynamically loaded and unloaded from an application.
- Platform and device specific assets can be loaded without having to download or store redundant assets for other platforms or resolutions.
- Downloading and installing the assets can be adjusted depending on the user's location, language or preferences.
- Applications can be fixed, changed or updated with new content without having to reinstall the application in the end-users' device.

1.2. Content formats for immersive displays

1.2.1. Omnidirectional videos

Omnidirectional video, also known as 360-degree video, are videos where a view in all directions is recorded at the same time. During playback, the viewer has control of the viewing direction. This control can be done via the mouse or the keyboard if the video is played in a computer or by head movement if the video is played in a HMD.

This format is produced by recording with a special rig of multiple cameras or with a dedicated VR camera that contains multiple lenses embedded in the device. The result of each camera is then stitched[3] to form a single video, in the first case (rig of multiple cameras) this process is done with a specialized video editing software. The synchronization of all the cameras is necessary for a best quality result in the stitching. An example of this kind of software is VideoStitch. This software uses three different techniques for synchronization [17]: Audio-based, Motion-based and Flashed-based.

Fig. 1.1 shows the different steps a content producer has to accomplish in order to create an omnidirectional video with VideoStitch:

[3] Process of combining multiple images with overlapping fields of view to produce a segmented panorama.

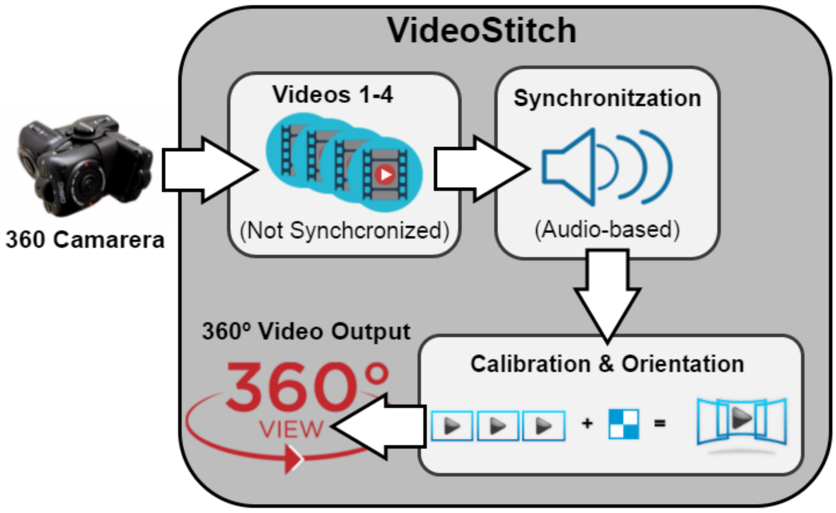


Fig. 1.1. Omnidirectional video creation process

During the development of this work a ring of four QBIC MS-1 cameras along with the VideoStitch software has been used to create omnidirectional videos. Once the recording time has ended the next step is to import the four videos into VideoStitch and synchronize them if needed.

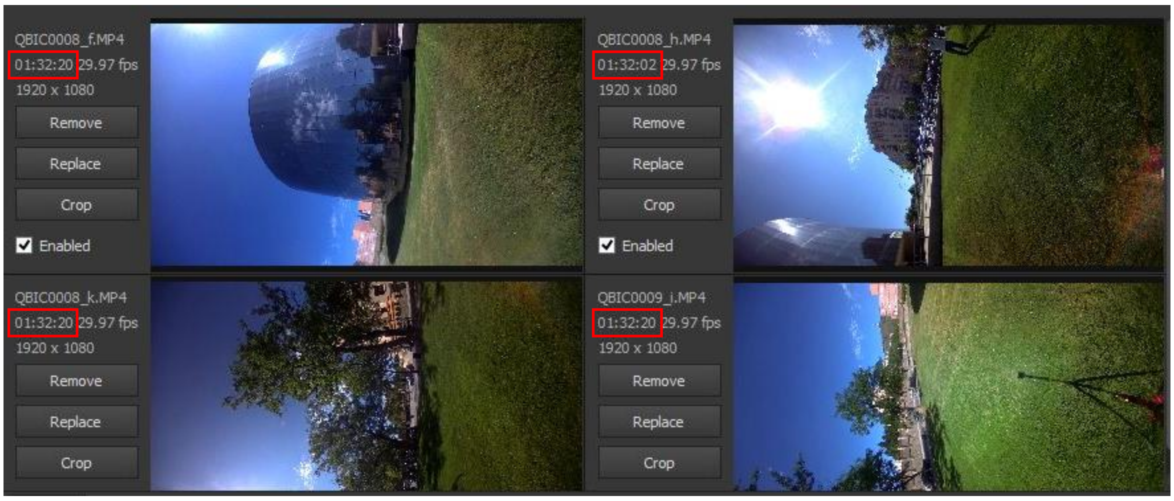


Fig. 1.2 VideoStitch synchronization process

Taking a look at Fig. 1.2, the red boxes show the recorded time of each camera, in a perfect situation these times have to be the same, however they aren't forcing the synchronization. In this example, a clap was done so Audio-based synchronization could be executed. Once the synchronization is perfect VideoStitch will stitch the four videos with the calibration option.

The next step is to fix the orientation of the calibration output. As seen in Fig. 1.3 the resulting orientation is wrong:

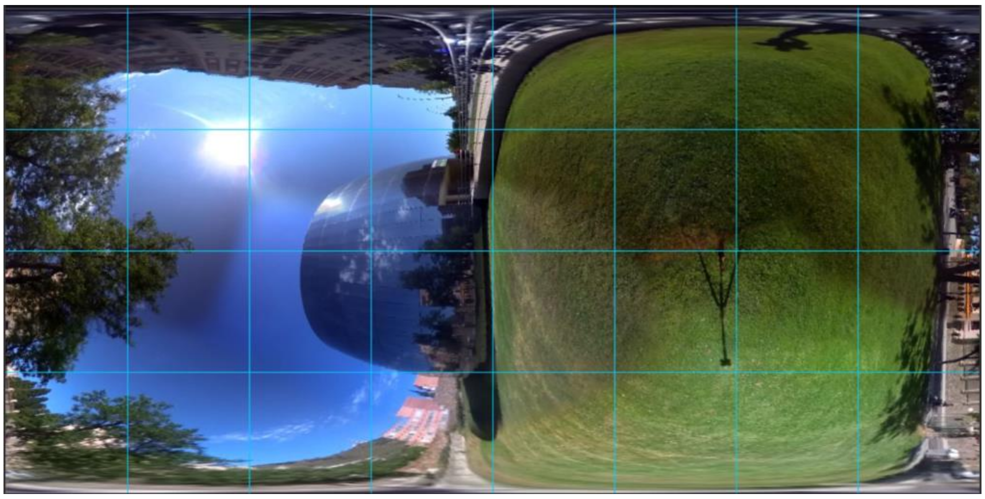


Fig. 1.3. Wrong orientation of the calibrated output

With the help of the mouse, the orientation is edited so when the experience played inside a HMD is coherent and undistorted. A good orientation of the omnidirectional video is seen in Fig. 1.4:

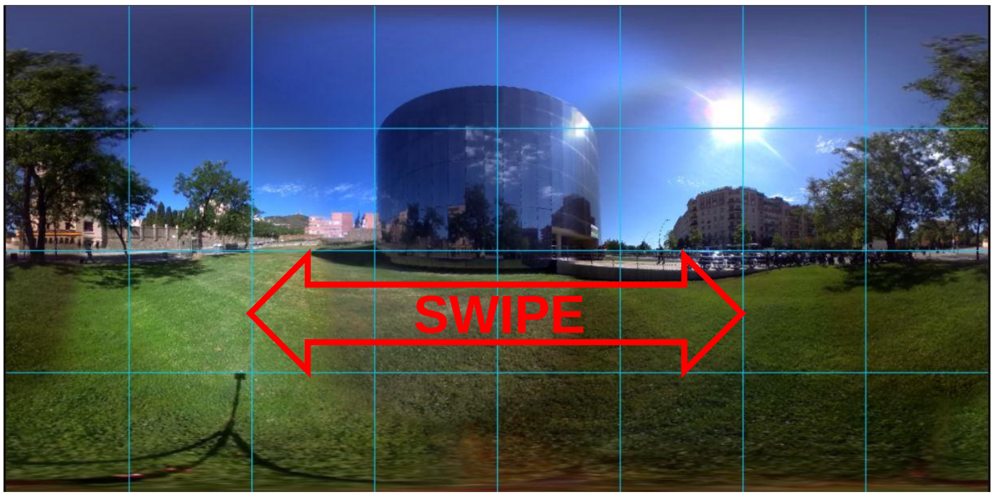


Fig. 1.4. Correct orientation of the calibrated output

In the second case (specialized VR camera) the camera itself does the stitching. After all the stitching is done and the panorama is created, with the use of UV[4] mapping[5] techniques this panorama will be projected inside a 3D sphere.

[4] The letters "U" and "V" denote the axes of the 2D texture because "X", "Y" and "Z" are already used to denote the axes of the 3D object in model space.

[5] 3D modelling process of projecting a 2D image to a 3D model's surface for texture mapping.

Comparing the two ways of producing omnidirectional videos, dedicated omnidirectional cameras are becoming the best economic way as you save the need to buy multiple cameras and use an external software for stitching.

Advantages and drawbacks

The advantages of this content format is the speed of production, and minimal requirements: in its minimal expression it only requires one omnidirectional camera for shooting and a virtual reality headset to view the obtained result. This allows creating rapidly simple virtual reality experiences for user consumption. Despite the production speed advantage this is at the expense of the experience quality. Using omnidirectional video imposes that the only movement preserving the place illusion is to turn your head around while being placed in the centre of a sphere exactly aligned with a vertical axis. Due to this limited sensorimotor contingencies the user will feel a smaller feeling of place illusion.

1.2.2. Traditional CGI

Traditional CGI (Computer-generated imagery) is content created by computer. This content is defined by the creation of 3D models typically used in the video game and film industry as well as in VR academic research.

In order to produce it you need to use 3D graphic, modelling and rendering software like Autodesk 3ds Max along with a game engine like Unity3D for example.

This content creation falls into three basic phases:

- 3D modelling - the process of creating a computer model from an object’s shape.
- Layout and animation - the placement and movement of objects and characters in the scene.
- 3D rendering - converts the model into an image by computer calculations based on light placement, surface type and other qualities. This is often performed in a mix of pre-calculated results and real-time calculations to adapt to the user’s input.

In Fig. 1.5 we can see the traditional CGI model creation process of a German shepherd from the game Call of Duty:

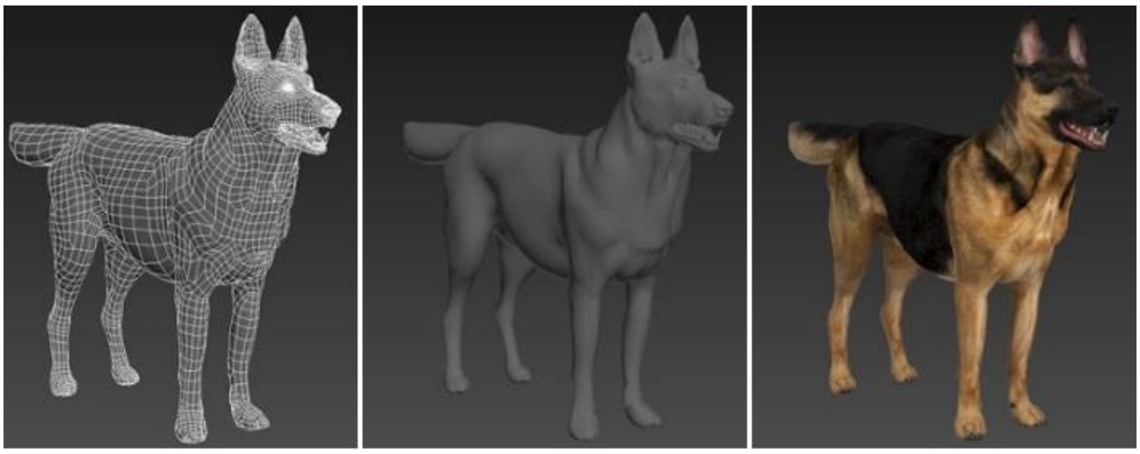


Fig. 1.5 Traditional CGI content creation process. Taken from [18]

In order to create a virtual reality experience with traditional CGI, it is necessary to use a head mounted display (HMD) like the HTC Vive [19]. In this case, since all the content is created by hand the time spent is greater, however scans of 3D physical objects in combination with hand-crafted animations and motion-capture data are increasingly used, which reduces the time.

Advantages and drawbacks

The advantages of this content is that the PI of the user inside the experience tends to be greater. Unlike the omnidirectional video, traditional CGI offers an immersive virtual environment where the user can move around, which will produce a greater sense of 'being there' (PI). Users using a HMD can move freely to the limit of the tracking space being covered. Allowing free movements is key to increase the PI of the user. Although traditional CGI content has many advantages, it also has a big drawback: this kind of content is very slow to produce. In order to create a decent environment for a VR experience you need experts in many sectors and the most important, time. In some cases the time and resources spent are often economically not viable.

1.2.3. Sensor mesh-based content

Sensor mesh-based content is capturing fast and accurate depth information of static objects and environments with a sensor.

To produce this content a depth sensor is needed. The number of this kind of sensors in the market nowadays is increasing, a few examples of this sensors are Structure sensor and Kinect 2. The first example has to be attached to an iPad however both use the same software in order to visualize the depth information sent by the sensors, Skanect 3D scanning software [20]. Along with the depth and infrared cameras, both sensors use a colour camera to capture colour (Structure sensor uses the iPad's camera). This implies that in order to create accurate 3D coloured models all cameras, depth, infrared and colour have to be calibrated.

In this work sensor-generated models are created with the Structure sensor. In Fig. 1.6 we can see how with the help of an iPad Pro 4 the Structure sensor scans the environment. The way this sensors work is by projecting a unique infrared pattern of dots out in front of it, then the infrared camera uses this pattern of dots to visualize the shape and distance of objects. The projection is done with a dedicated infrared laser projector. Each colour of the image determines the distance to the sensor; red parts are closer, blue parts are further:

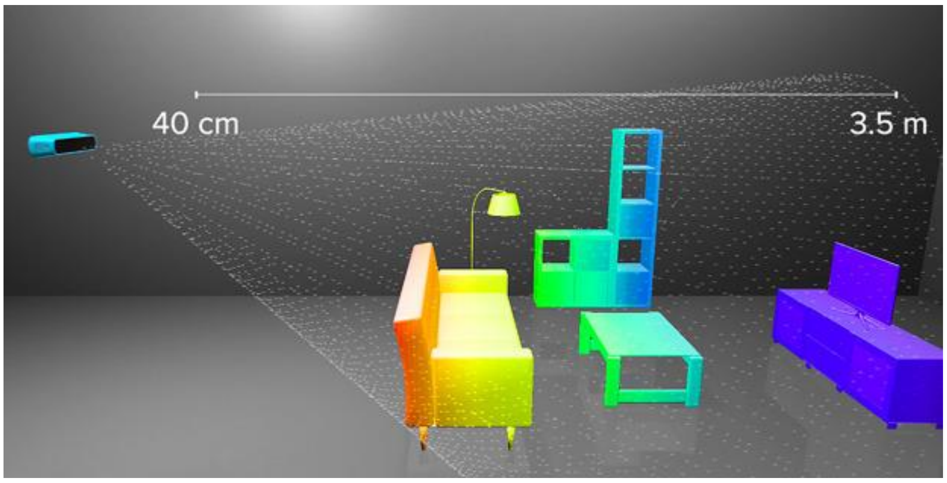


Fig. 1.6 Structure sensor scanning. Taken from [21]

The information obtained by the sensor is send through the Wi-Fi interface to the Skanect software. Skanect will create a 3D model without colour from this information. Next step is to process this model where many options are possible from; hole filling, remove parts and colorize. In Fig. 1.7 we can observe an example of 3D model before and after been coloured:

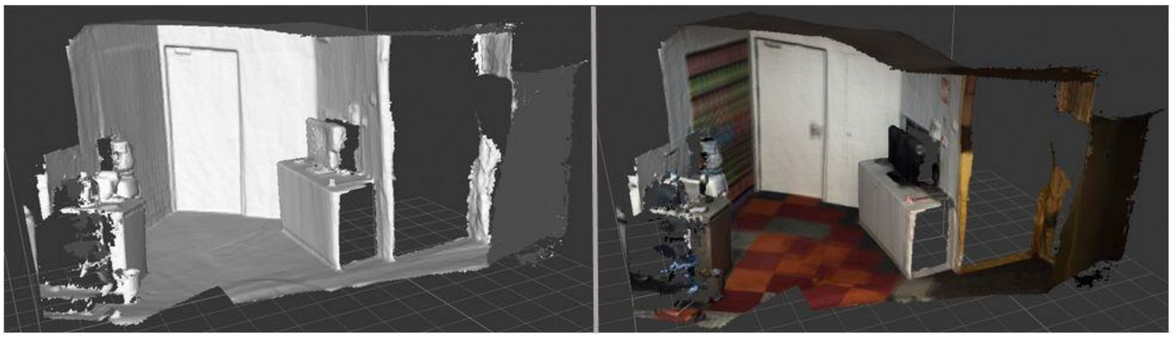


Fig. 1.7 3D model before and after been coloured

Advantages and drawbacks

The advantages of the sensor mesh based content are similar to the traditional CGI, however because it depends on the hardware, the quality of the results obtained from the sensors is worst nowadays. Nevertheless 3D models obtained from the sensors can be edited in order to increase the quality of these and in this way increase the speed of production. New sensors, more powerful and precise, will be developed in the upcoming future and this kind of content creation will outpoint in some part the traditional CGI.

In terms of user experience, being able to scan and then track the environment with the real world gives the user the possibility to interact inside the VR experience as if it was the real world increasing in some way the PI and Psi of the user. The main drawback is the need of hardware. This kind of hardware have limitations, and nowadays it is difficult to scan objects or environments with small details because of the range of precision of the sensors. Other problems to overcome is not all objects can be scanned, too bright or transparent objects are difficult to scan or sometimes impossible.

1.2.4. Volumetric Video

Volumetric video content is the introduction of a 3rd dimension inside a normal 2D video captured scene in order to give some “volume” to the scene.

This content is produced by attaching a depth sensor, in this case a Kinect 2, to a DSLR camera (there is no need to use an external video camera as the Kinect itself has a colour camera embedded, however it is recommended for high quality content). Both cameras have to be calibrated, after this process is done the relationship between the two of them have to be calibrated too in order to properly project the colour information from the video camera onto the depth. To do this, you will need to determine the physical position of the two cameras relative to one another. This will allow to combine the two data streams (colour and depth) into one 3D scene [22].

In Fig. 1.8 we can observe the 3D mesh information of the Kinect 2 in conjunction with the RGB information of the colour camera:

All this processes can be done by a non-free licensed software (currently in closed beta) named DepthKit [23] with Unity3D. However there is another option using Unity3D and a free (only for research reasons) asset named Kinect v2 Examples with MS-SDK [24]. Both options use the Kinect for Windows SDK. Like the sensor mesh based content, the Kinect uses the infrared camera in order to determine the shape and distance of objects.

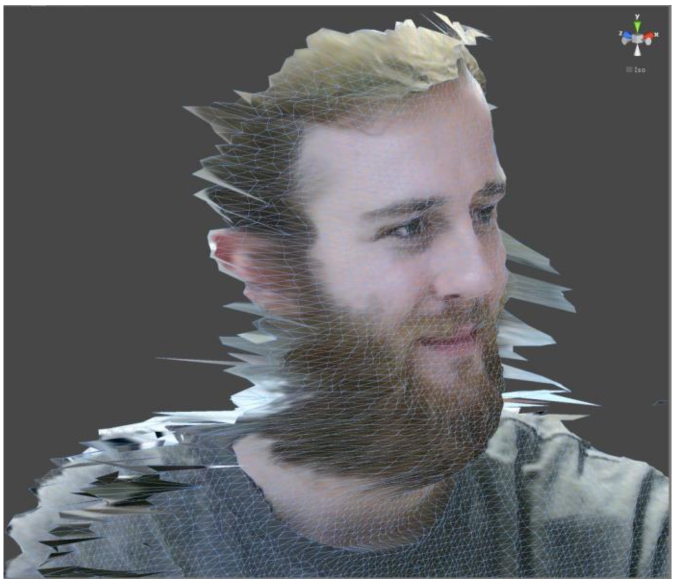


Fig. 1.8 Depth info with RGB

Fig. 1.9 shows a capture from different perspectives in order to see the volume of the model created from the Kinect 2:

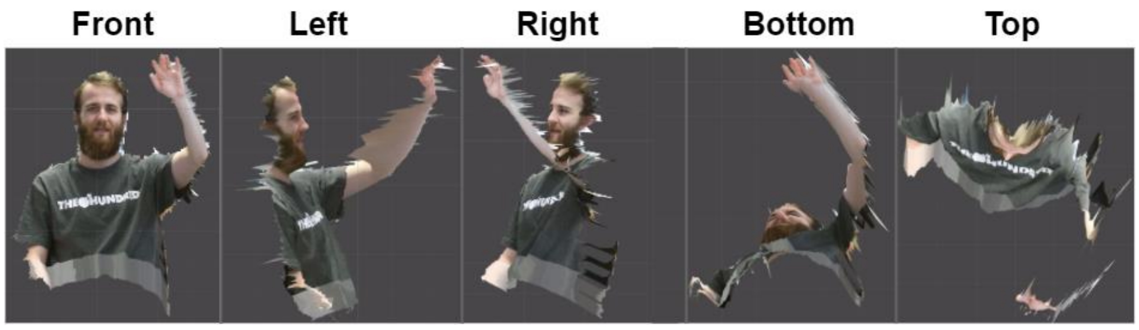


Fig. 1.9 Volumetric video perspective snapshots

Advantages and drawbacks

The main advantage of the volumetric video content, in comparison with the sensor mesh based content, is the ability to capture 3D content with movement. Furthermore, this content can be captured live with the possibility to combine it with the other three contents mentioned earlier. Despite these advantages, the main drawback of this content is that the Kinect's depth sensor only captures inside its Field of view[6] (FoV) thus the 3D model will be incomplete out of this,

[6] Extension of the observable world at any given time.

however there’s the possibility to combine several Kinects in order to create a full 3D model although this involves the development of complex software.

In Fig. 1.10 we can observe how the hand blocks the FoV of the Kinect 2 creating an incomplete 3D model. As mentioned before, in order to create a full 3D model, more Kinects will have to be used in different positions.

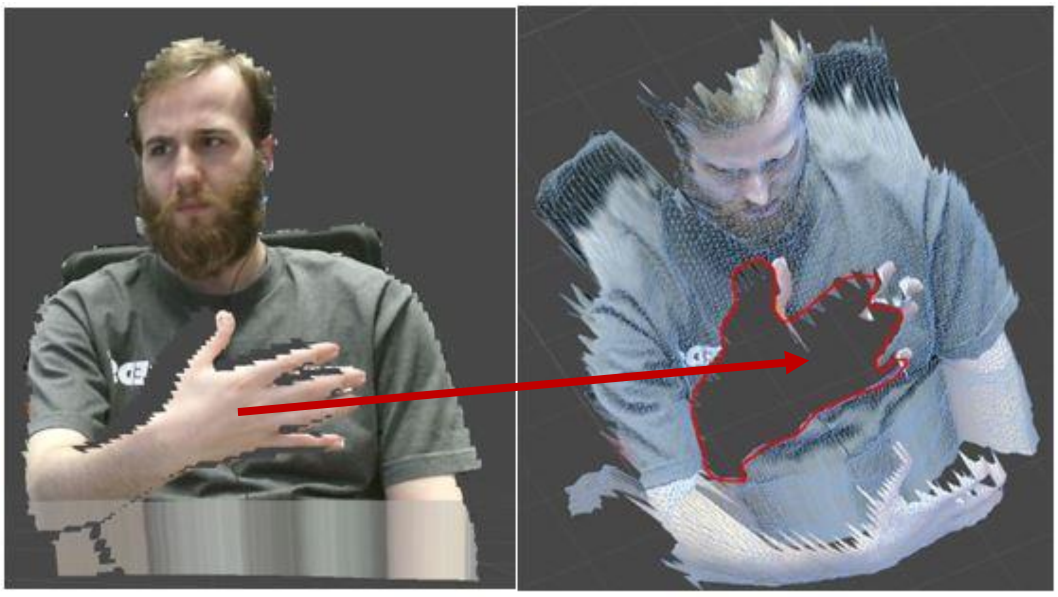


Fig. 1.10 Kinect 2 FoV block

In the case of using only one Kinect, the PI of the user is bigger comparing it with the omnidirectional video due to the factor that within a HMD the user has the possibility to lean out a bit and check the volume unlike the omnidirectional video which everything is 2D. This option is not possible inside a smartphone within a virtual head due to movement limitations, only 360º head turn is permitted.

In terms of speed of production, volumetric video is fast to produce in live conditions as it only needs to have the Kinect sensor connected and a platform to be reproduced on. However if this content need to be recorded and reproduced after you need to use external software like DepthKit or develop your own one, the main inconvenient of this last one is finding the way to save the depth and colour information for later playback.

In this work for volumetric video playback (no live conditions) a script was created based on the Kinect v2 Examples with MS-SDK asset in order to save the colour and depth information (a JPEG image and two files with information of depth and colour in a raw format[7]) of each frame (scripts for volumetric video creation can be found in the Annex 2: Code). The main problem of this method is the amount of space needed for a few seconds; 25 frames (with a resolution of 1920x1080)

[7] Data format not yet been processed and therefore containing all the data from the original file.

are equivalent to 239MB. We have to take in account that we have to safe the whole information of the image in order to create the coloured mesh.

The calculations are:

- 25 raw colour data 1920x1080 in RGBA32 8 bits per channel.
- 25 raw depth data 512x424 in ARGB4444 16 bits per pixel.
- 25 JPEG images (more or less 1.2MB)

$$239\text{MB} = 25\ \text{frames} \cdot \left(\frac{((1920 \cdot 1080)\ \text{px} \cdot 4\text{B} + (512 \cdot\ 424)\ \text{px} \cdot 2\text{B})}{1024 \cdot 1024} + 1.2\text{MB}\right)$$

CHAPTER 2. HARDWARE & SOFTWARE TECHNOLOGIES USED

2.1. Hardware

The platform developed in this work for innovative VR content distribution needs hardware in order to create and consume this content. In this section an example of this hardware and its specifications will be described. A more detailed description of these technologies can be found in [25].

2.1.1 Google Cardboard

The Google Cardboard glasses [26] (GC) are a low cost (14.99 USD) open source VR viewer developed by Google. This display is phone based, in other words, you have to place your smartphone in front of the lens in order to enjoy the VR experience. This viewer is made from a piece of cardboard cut into a precise shape (all the specification on how to make you own cardboard display can be downloaded from Google's website), 45 mm focal length lenses, a conductive lever (triggers a touch event on a phone's), Velcro and a rubber band for a total of 85 grams without the phone. This gear can be used for any smartphone with a screen up to 6 inches (150 mm).

The way this display works is it uses the smartphone's gyroscope to track the head rotation, this way the users feel the sense of being immersed inside the VR experience. The GC uses the phone's specifications related to VR as motion-to-photon latency[8]. The high latency that nowadays many smartphone's offer (80-100ms), can cause motion sickness on users that experiment with long time VR content. GC solves this problem, partially, forcing users to hand-held the display as this slows down the speed at which the head is turned, nevertheless VR content needs a motion-to-photon latency of <20ms for users to enjoy the experience.

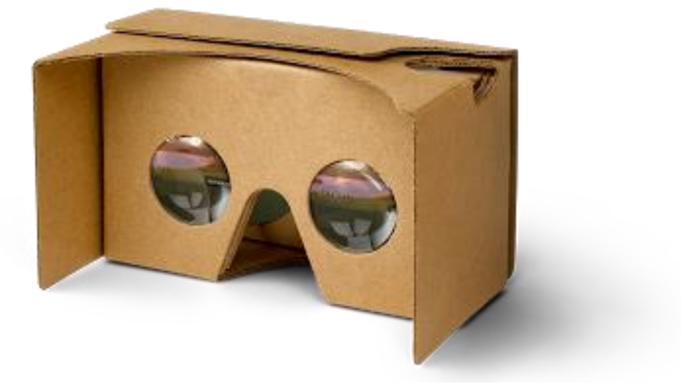


Fig. 2.1 Google Cardboard glasses. Taken from [27]

[8] Time needed for a user`s movement to be fully reflected on a display screen.

2.1.2 Samsung Gear VR

The Samsung Gear VR [28] is a mobile virtual reality headset developed by Samsung Electronics in collaboration with Oculus with a price of 99€. Unlike GC, the Samsung Gear VR only works with the following Samsung Galaxy smartphones: S6, S6 Edge, S6 Edge+, Note 5, S7 and S7 Edge. However this display has a custom inertial measurement unit (IMU) for rotational tracking connected to the smartphone via micro-USB. This IMU is more accurate and well calibrated with low motion-to-photon latency (<20ms) than the internal smartphone IMUs, used by the GC, which will reduce motion sickness significantly. The Gear VR headset has a FoV of 96º (nominal), it also includes a touchpad and a back button on the side, as well as a proximity sensor to detect when the headset is on, all together weights <400 grams without the phone.

With everything discussed above, despite the GC and the Gear VR are both phone-based displays this last one offers a better experience due to the drop in the motion-to-photon latency.

Fig. 2.2 Samsung Gear VR. Taken from [29]

2.1.3 HTC Vive

The HTC Vive is a VR head mounted display (HMD) developed by HTC and Valve corporations with a price of 899€. The HTC Vive comes together with 2 base stations (for user tracking), two hand controllers along with all the components related to connect all these together with the PC. This display needs some minimum PC requirements like NVIDIA GeForce GTX 970 or greater, Intel Core i5-4590 equivalent or greater, 4GB+ of RAM, compatible HDMI 1.3 video output and 1x USB 2.0 port to work properly. This is a difference in comparison with the phone-based displays, however this will result in a better VR experience as it offers 90 Hz of refresh rate, a FoV of 110º (nominal) and OLED displays of 1080x1200 per eye.

The HTC Vive, like all the HMD, has a user tracking system. The display uses more than 70 sensors including a MEMS gyroscope, accelerometer and laser position sensors. If both base stations are used to track the user's movement, it is said that the HTC Vive operates in a tracking space of 4.6m^2.

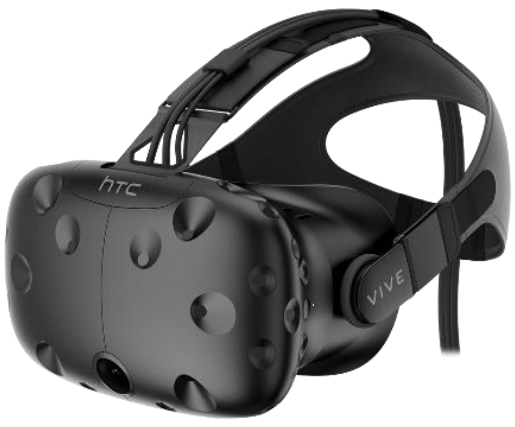


Fig. 2.3 HTC Vive. Taken from [30]

2.1.4 QBIC MS-1 (Omnidirectional camera)

For video capture omnidirectional cameras were used. The omnidirectional camera consists in a rack of 4 QBIC MS-1 [31] cameras with 185º of horizontal angle view. These cameras can record in a maximum resolution of 1080p60 and can be accessed through an APP via Wi-Fi so the user can execute play on the four cameras at the same time (with some milliseconds of difference).

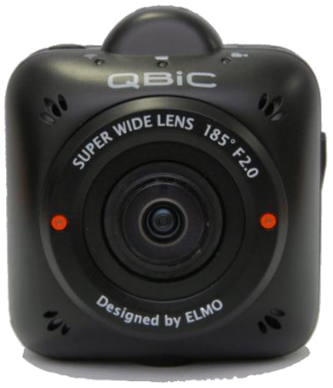


Fig. 2.1 Camera QBIC MS-1. Taken from [32]

The omnidirectional camera has to be calibrated previously so the obtained result is the desired one. Before calibration, synchronization is very important, if synchronization fails, the output video will have doubled images during playback at different moments. This is due to the cameras have a FoV of 185º, if people move too close around the cameras there is a big possibility they enter in the FoV of more than one camera. If synchronization fails, when stitching the videos you will have two different frames stitched together which will cause a ‘ghost’ effect.

When calibrating, objects near the camera have to be at a certain distance, 3m is the recommended so you don’t get any parallax effect.

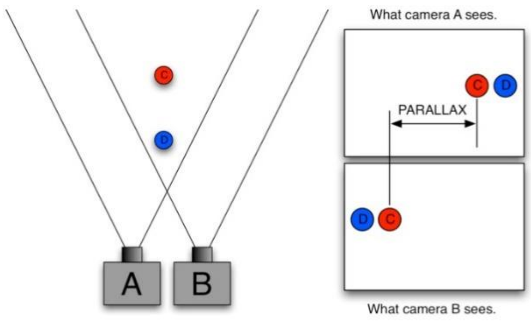


Fig. 2.2 Parallax effect diagram. Taken from [33]

This effect is caused due to the four cameras do not have the same point of view. It’s the same thing you can experiment by putting a finger very close between your eyes; you won’t be able to see it correctly as it will be too close to get a correct complete image of it. If you alternatively close one eye, you will see how the finger changes its position, the distance between this two positions is what is called parallax effect. If you move the finger further this distance will decrease.

When stitching the videos together, if objects have to be near the cameras better try to have the object only on one camera lens FoV, in other case the parallax effect will be seen. Despite everything you try to do, removing all the parallax is impossible, however there are ways to reduce it. In Fig 2.6 we can observer an example of parallax error when stitching the videos.

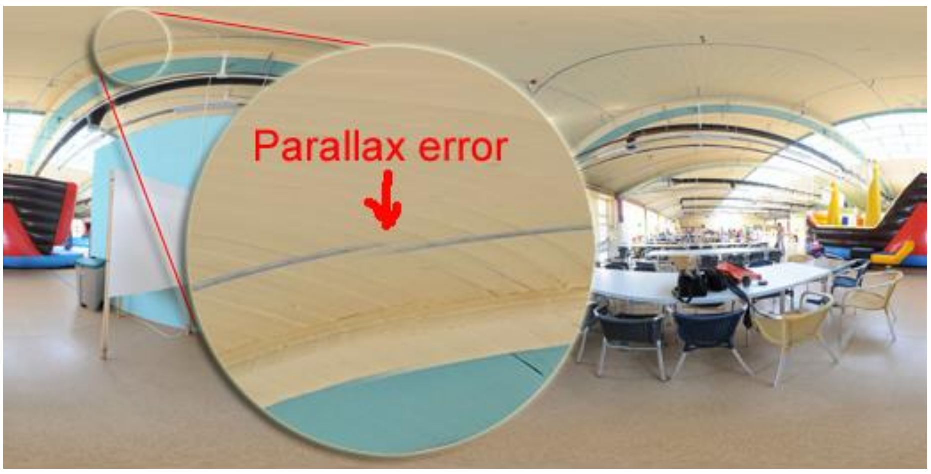


Fig. 2.6 Parallax effect error example in omnidirectional video. Taken from [34]

2.1.5 Structure sensor

The Structure sensor [35], developed by Occipital, is a 3D scanner that has to be attached to an iPad, in the case of this work an iPad Pro 4. Occipital also provides, through the Apple store, several apps to use with the Structure sensor. In this work we will use three of them: Calibrator (Used to calibrate the Structure sensor's infrared camera with the iPad's colour camera), Scanner (Used to scan individual objects) and Room Capture (Used to scan small spaces/rooms without many details).

The sensor has a maximum recommended range of 3.5m+ within a precision of 30mm (1%) and a minimum recommended range of 40cm within a precision of 0.5mm (0.15%). Its FoV is 58º horizontal and 45º vertical and it works in a framerate of 30/60 frames per second. Once the Structure scanner is used to scan, the resulted 3D model can be sent through the iPad's Wi-Fi interface to Skanect for further processing.

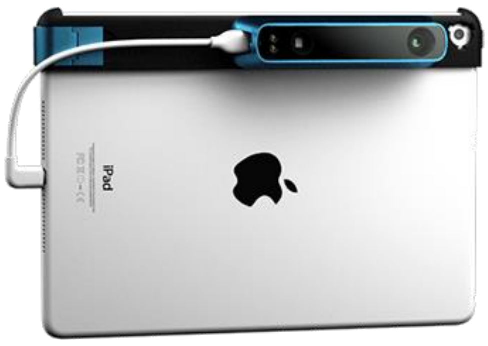


Fig. 2.3 Structure sensor with iPad Pro 4. Taken from [36]

2.1.6 Kinect 2

The Kinect 2 [37], developed by Microsoft, is a sensor which combines three type of cameras: Colour (resolution of 1080p at 30 Hz), Infrared (resolution of 512x424 at 30 Hz) and Depth (resolution of 512x424 at 30 Hz with a FoV of 70x60 within a range of 0.5-4.5 meters). These three cameras combined together can provide 3D models at real time. In this work we will use Kinect's SDK together with the Unity3D asset Kinect 2 Examples with MS-SDK in order to record volumetric video.

In section 4.2 we will describe how the Kinect 2 was used in order to track the user's body inside a VR environment.

Fig. 2.4 Kinect 2. Taken from [38]

2.2. Software tools

2.2.1. Docker

Docker [39] is an open source project that automates the deployment of applications inside software containers. Docker containers, because it wraps up the application in a complete filesystem containing all the necessary parts needed to run, guarantees that regardless environment changes it will always run the same.

In this work Docker was used in order to have all the elements that composed the web application wrapped up in containers and all working in the same operating system, in this case Linux. This removed the problem of having to develop specifically for each OS (this was one of the main problems at the beginning). As mentioned before, this software tool guaranteed that no matter what environment the web application was to be deployed in, it would work making it easy to test in different setups.

The web application developed in this work was divided into four different containers, although at the beginning having this many containers was tedious to configure, other Docker tools were used during the development in order to automate this: Docker Compose and Docker Hub. Later, in section 3.3 of this document the reasons why this tools were used will be explained.

2.2.2. GStreamer

GStreamer [40] is a free and open-sourced multimedia framework software based in a pipeline structure. For this case a pipeline is referred as a set of data processing elements connected in series where the output of one element is the input of the next one. GStreamer can read media files in one format, process them and export this media files in another.

GStreamer was used in the first stage of development in order to create a stream with several media files through a RTSP pipeline. The pipeline was launched with the GStreamer RTSP server in order to create a low-latency streaming. Other reasons why GStreamer was used in the first stage of development were audio and video streams had to be exported to a specific format so the Unity3D player could play the content. Depending on what content format the editor worked with, the web application had to create a specific pipeline so the output was the desired one. Furthermore GStreamer also has support for all OS.

2.3. Web technologies

For the development of the web application different web technologies have been used. In this section each technology used will be described and justified.

2.3.1. Node.js + Express

Node.js [41] is an open-source, cross-platform JavaScript runtime environment for developing tools and applications. Node.js is based in an event-drive architecture[9] capable of asynchronous input/output processing.

On the other hand, Express [42] is an open-source web application framework for Node.js. It is designed for building web applications and APIs. It is the standard server framework for Node.js.

These two technologies were chosen in order to create the core of the web application. Other reasons were how Node.js works with its package manager *npm.* Npm consists in a command line client that interacts with a remote registry that allows users to consume and distribute JavaScript modules that are available on the registry. This is a fast and easy way to add different technologies into the web applications. Last but not least, during the whole development of the work the JavaScript IDE named WebStorm was used. WebStorm has the option of creating a new Node.js + Express project which facilitated the start of the work by creating the main structure of it.

2.3.2. Angular

Angular [43] is an open-source JavaScript front-end web application framework. This framework created by Google was designed to make front-end development as easy as possible.

Angular was chosen due to it combined perfectly with Node.js and Express. Although Angular combines also with MongoDB in order to create the MEAN

[9] Software architecture pattern promoting the production, detection, consumption of, and reaction to events.

stack, later on the reasons why Redis was chosen over mongoDB will be described. Angular was also chosen because it was fast to incorporate and easy to learn in order to create a not very complex web application, however Angular has plenty of options and needs time in order to be an expert.

2.3.3. Bootstrap

Bootstrap [44] is an open-source front-end web framework for designing web applications. Bootstrap contains HTML and CSS based design templates for many interface components in order to give a modern appearance to the web application without investing any time in programing the visual style.

Bootstrap was chosen for many reasons, save programing time as well as it is designed to give responsiveness[10] to the web application. Last but not least, bootstrap is easy and fast to incorporate into the web application at any development stage.

2.3.4. Redis

Redis [45] is a NoSQL open source key-value database. You can compare it as a giant array in memory for storing data, data that can be strings, hashes, data sets or lists. However, Redis does not permit queries, you are only allowed to insert and obtain data beside the common operations on sets (difference, union and insertion).

Redis was chosen for several reasons. One of the main reasons was how fast it was to implement in the web application. Without any big changes in the code structure, the Redis database was added. This was the main reason why Redis was chosen above mongoDB which needed a more complex configuration. Redis was also chosen due to the data model the web application used, projects (keys) were connected to other details all grouped in a JSON (value).

[10] Adaptation of the web application's layout to the size of the device where it's viewed.

CHAPTER 3. EVOLUTION OF THE DESIGN OF THE CONTENT DISTRIBUTION PLATFORM

As mentioned before, VR headsets are becoming very popular nowadays and this has provoked new innovative content to emerge. The problem is, there is no dedicated platform for user to consume and enjoy the experience of this VR content today.

In March 2015 YouTube was one of the first introducing the possibility for users to upload and playback omnidirectional videos followed by Facebook in September of the same year [46]. Despite this new feature, both applications don’t aim in distributing VR content. However VRideo [47] and VRAPP [48] are two examples of platforms aiming to distribute omnidirectional videos. Nevertheless these platforms in conjunction with YouTube and Facebook only distribute 360º videos, unlike the platform developed in this work.

The main goal of this work is to develop a web platform for the distribution of innovative VR content based in omnidirectional video and sensor-generated mesh. In order to satisfy as many users as possible, content (based on omnidirectional video) exported from Adobe Premiere was converted to MPEG-DASH so it can be reproduced in as many different smartphones as possible regardless of internet connectivity or device resolution. Users have the option to publish the content they want to view and select it from a custom Unity3D player created for the ImmersiaTV project. On the other hand, content exported from Unity3D as AssetBundles (based on sensor-generated mesh) can also be published but has to be viewed through a Unity3D player developed in this work.

The web application was developed in four different stages. This section will describe what technologies were used in each development stage and why. Each stage will begin with a brief description of the motivation followed by the architecture. The architecture of each development stage is very important as it describes the function of all the new “pieces” (new technologies) and where they fit in the “puzzle” (distribution platform). A list of problems encountered is described and the section ends with a user experience example.

Finally the chapter will conclude with a brief explanation of the development conclusions.

3.1. RTSP-based distribution

The idea of this first version of the web application was to create a VR content streaming service using the RTSP server of GStreamer. Potential users using a custom Unity3D player would have the opportunity to connect to the stream and visualize the content. The first development stage of the web application started by choosing the technologies that were going to be used.

3.1.1. Architecture

The core of the web application was developed with Node.js and Express in conjunction with Angular. For the visual part of the web page Bootstrap was used. This four elements describe in section 2.3 will be common during the development of the web application. In order to list the media content for user selection a module named jsTree was used. This module allowed to create a "mirror" of a folder and its content and visualize it in a "tree" layout as seen in Fig. 3.2.

As mentioned before this first stage was based in a RTSP distribution using a GStreamer pipeline. This pipeline was generated automatically depending on the selected media content's format.

The architecture of the RTSP-based distribution web app is show in Fig. 3.1:

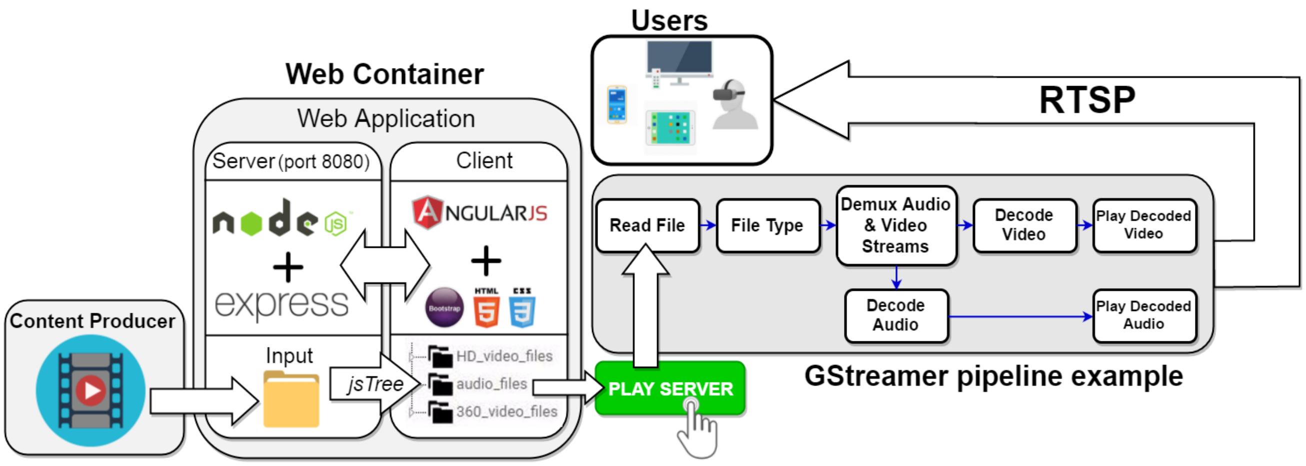


Fig. 3.1 RTSP-based distribution web app architecture

Media content producers upload their content to the Input folder. Users then select this content and start the server. The GStreamer software will create a pipeline and the Unity3D players will connect to the pipeline in order to view the content of the stream created via RTSP.

3.1.2. Problems encountered

The main problem of the RTSP-based distribution web application was that once the user had created the pipeline with the content to stream no more streams could be created due to GStreamer limitations: it could only create one pipeline per socket. If the user wanted to view various video streams, all the media content had to be selected of all the different streams, and finally select in each player the different ids of the media contents of each streams.

Although the web application was developed for remote deployment in a server using Linux, local tests[11] were made in order to remove network condition constraints. The problem with these localhost tests was that the GStreamer software is different for each operating system. Furthermore, the web app used processes to create the pipelines and each OS terminal works in a different way. This created the need of different scripts for each OS which increased the development time and application size.

Last but not least, because the distribution of the media streams was through a GStreamer pipeline created by a process, if the sever crashed no more content could be served. This was a key problem to be solved in future releases.

3.1.3. User experience

The view of the first version of the web page was as seen in Fig. 3.2:

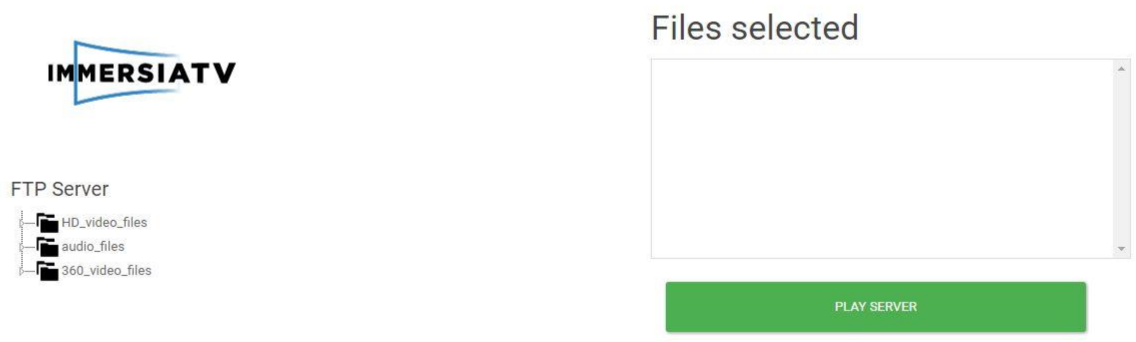


Fig 3.2 View of the RTSP-based distribution web page

The web page view was divided into two sections. The first one (left hand side), FTP Server, is the mirrored server folder containing all the media content. The second (right hand side), Files selected, is where the selected media content is listed.

In order to create a stream, a potential user must select all the content he wants to introduce in the GStreamer pipeline. In this example the content inside the folder *HD_video_files* named *HD_video_example.h264* is selected as shown in Fig. 3.3:

[11] Tests carried out where communication between client and server is within a LAN.

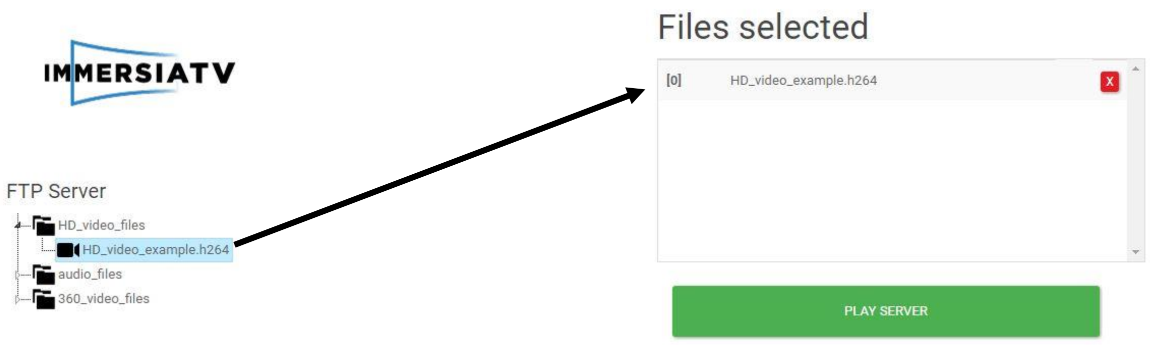
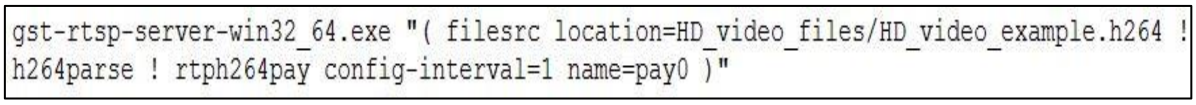


Fig 3.3 Media content selected

Each table entry is composed of three elements: id, media content name and a button for removing. The id (table index[12] of the selected element, in this example the first element in the table has index “0” so the id will be “0”) has to be introduced in the players in order to view the correct media content. Usually the players will be configured previously thus the order of selection will be crucial.

Once all the content has been selected the user can now press the *PLAY SERVER* button. GStreamer will create a new pipeline like the show in Fig. 3.4 and users will have the opportunity to connect to the stream.



```
gst-rtsp-server-win32_64.exe "( filesrc location=HD_video_files/HD_video_example.h264 !
h264parse ! rtph264pay config-interval=1 name=pay0 )"
```

Fig. 3.4 GStreamer pipeline example

In order to stop the stream, once the *PLAY SERVER* button is pressed this will change automatically to *STOP SERVER,* if the user presses this button not only the stream will stop but it will destroy the processes containing the GStreamer pipeline. This means that if the user starts the server again the media content will start from the beginning and not from where he had stopped before.

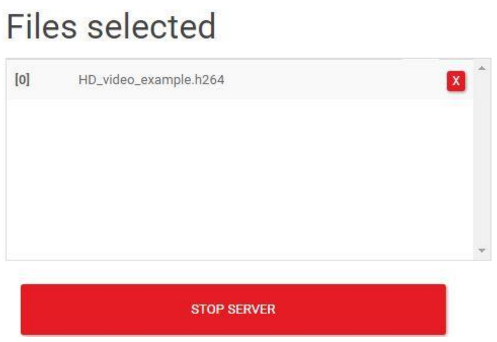


Fig. 3.5 Stop server option

[12] Position of an element inside a list starting with 0.

3.2. DASH-based distribution

The main purpose of this DASH-based distribution web app was to change the distribution technique in order to serve better quality content to us much possible users regardless on network constraints and device characteristics.

In this second stage of development of the web application, RTSP was changed to MPEG-DASH. This meant GStreamer was deleted from the web application. At this stage, the VR content users had to consume was exported from Adobe Premiere. This content was then converted to MPEGDASH using a script.

Users consume the content via a Unity3D player which reads a content JSON file in order to list the available content. This way users have a variety of contents in a more organized way.

As mentioned in the first stage of development, one of the biggest problems was trying to deploy the web application within different OS. The solution for this problem was to use the Docker tool. The reasons for the use of this technology have been stated in section 2.2.1.

3.2.1. Architecture

The core architecture of the web page remained the same (Node JS, Express, Angular and Bootstrap). However at this stage of development the web application was running inside a Docker container (Web app container Dockerfile and scripts can be found in the Annex 2: Code). As we mentioned earlier, Docker works over Linux so only one configuration was needed, this reduced substantially the amount of code as all the rest of OS configurations, except for Linux, were removed. In addition, a separate container for the DASH conversion (DASH container Dockerfile and script can be found in the Annex 2: Code) process was created. From now on, every new technology added to the web application architecture was introduced in a Docker container in order to have everything wrapped up in an organised environment.

As in the first stage, the web app mirrored two folders using the module jsTree, one for the exported Premiere projects (Input) and one for the DASH converted content (Output). This folders were created too in the Docker container using *data volumes.* This *data volumes* are copies of the folders inside the Docker file system, any change made in these folders will be done to the *data volumes* too.

Finally the users published the content in a contents JSON file that was to be consumed by a Unity3D player.

The architecture of the DASH based distribution can be seen in Fig. 3.6:

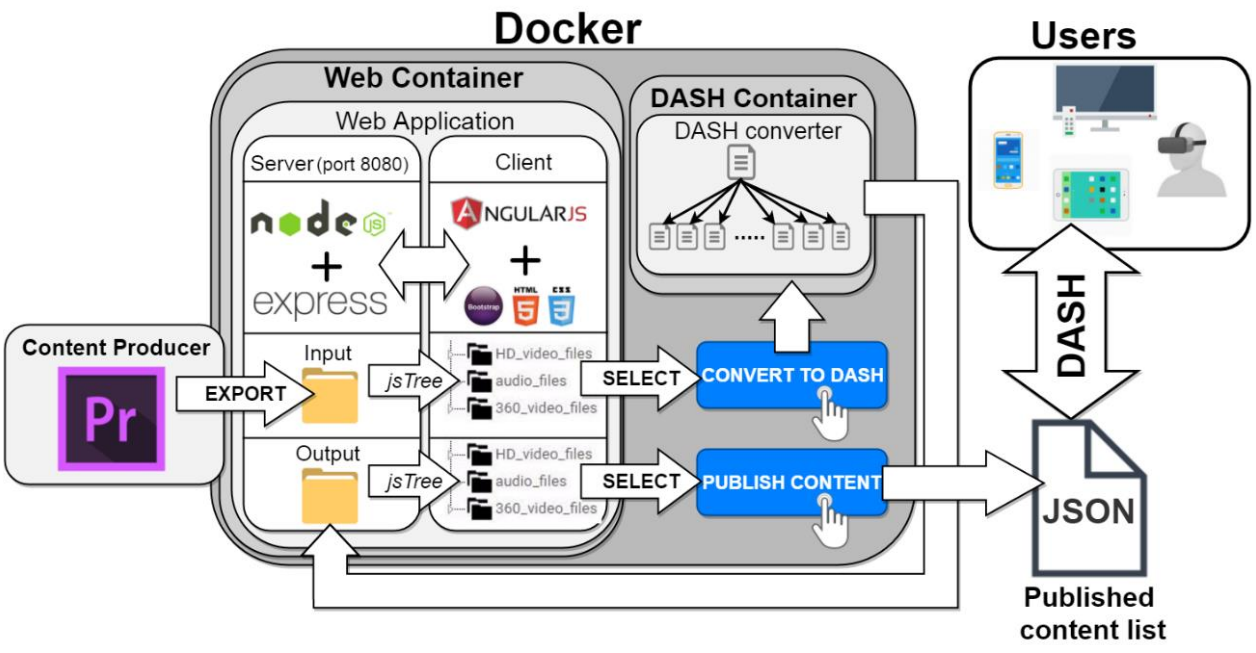


Fig 3.6 DASH based distribution web app architecture

Despite the complexity of the architecture at this stage of development, due to the way the web application was organized made the workflow much easier for everybody. Furthermore, the web application became more robust.

3.2.2. Problems encountered

The main problem during this stage of development was how hard and complex was to configure each of the elements as there was no know way to automate this process. Each Docker had to be build and run every time a test was to be made, this was a massive waste of time. Another big problem was users had to refresh the web page in order to view new content added to the folders. Other problems were the web page's lack of information when process/errors occurred during the MPEG-DASH conversion. Due to the way the web application was programed at that stage, users could only convert one project at a time.

There were also problems related to the MPEG-DASH conversion when several projects had the same XMLs file name. The problem was the output folder from the MPEG-DASH conversion took the same name as the Premiere XML file, when trying to convert another Premiere project with that same name it could not be converted as the folder was already created.

Last but not least, as the web page at the moment didn't use any kind of database there was no way to know what processes where taking place at that moment, so users could try and convert to MPEG-DASH the same project thus generating an error.

3.2.3. User experience

The view of the second version of the web page was like seen in Fig. 3.7:

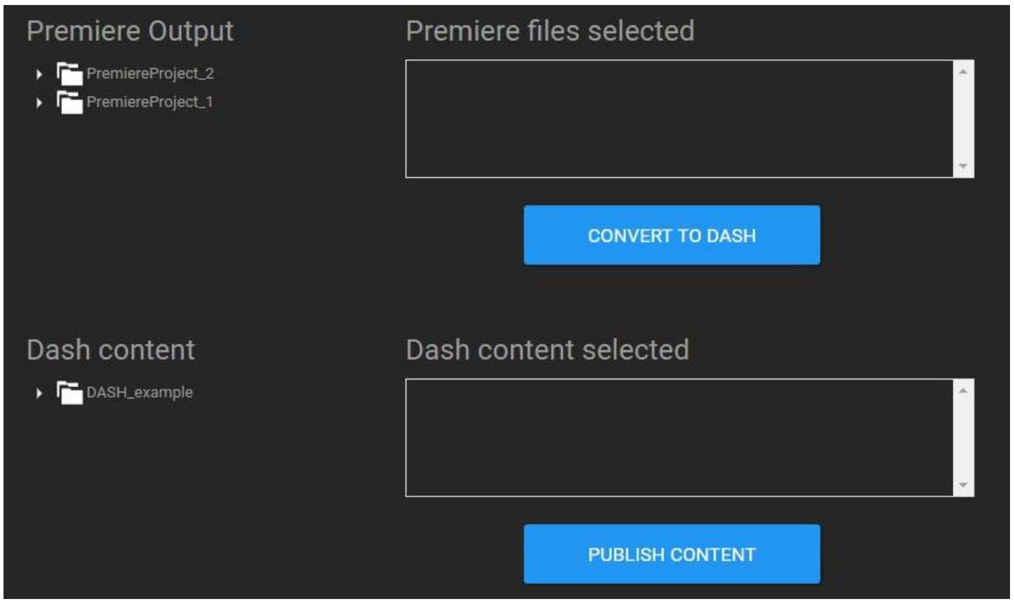


Fig 3.7 View of the DASH based distribution web page

The web page was divided into two sections, Premiere and DASH. Both sections had the same structure as the first version, mirrored folder and files selected table. In order to publish and consume new content exported from Premiere, users firstly had to select this content from the *Premiere Output* column and finally press the *CONVERT TO DASH* button.

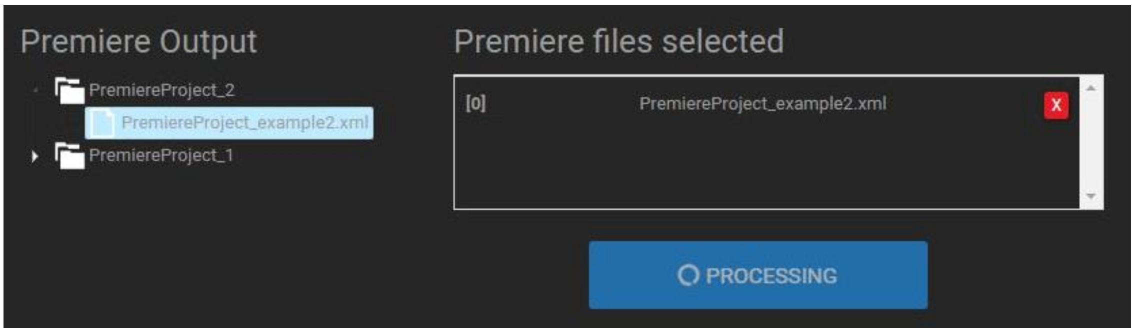


Fig. 3.8 Premiere to DASH conversion

As shown in Fig. 3.8, the button's text changed to *PROCESSING*, this is the only information the user had in order to know in which state is the DASH conversion. Once the DASH conversion ended, the user had to refresh the web page in order to view this new content in the DASH content column.

By selecting all the content to publish and pressing the *PUBLISH CONTENT* button, a JSON file was created as shown in Fig 3.9.

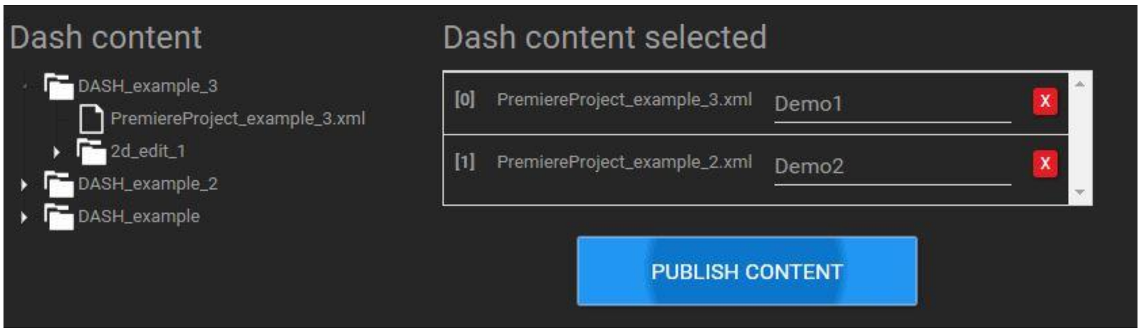

Dash content
DASH_example_3
PremiereProject_example_3.xml
2d_edit_1
DASH_example_2
DASH_example
Dash content selected
[0] PremiereProject_example_3.xml Demo1
[1] PremiereProject_example_2.xml Demo2
PUBLISH CONTENT

Fig. 3.9 DASH content publishing

This JSON file contained the ULR paths of each published XML file in order to list the available content in the Unity3D player.

Unlike in the first version where the user had to select the media files he wanted to stream through the GStreamer pipeline, in this second users select XMLs of metadata created from Adobe Premiere containing information of the media files of each consumable content (an example of this XML file can be found in the Annex 2: Code). This way users can publish multiple contents and selected what to view through the Unity3D player.

3.3. DASH-based distribution, v2.0

In the third stage of development the web page needed a big change in terms of design and user friendliness. At the moment users didn't have any information on the MPEG-DASH conversion state and had difficulties to find the content they wanted to convert or publish. A database was added, in order to save and visualize the state of conversion. The new design introduced many new functionalities like the possibility of converting several projects at the same time or searching and filtering the content. In general the web page appearance acquired a more professional look.

One of the biggest non solved problem in previous versions was the dependency on the web application in order to serve the media content. This problem was solved introducing a web server that was going to serve the media content separate from the web application, this way if the web application crashed the media content could still be available. Another thing that needed a big change was how all the services were started and deployed. At this stage all the different services had to be build and executed separately wasting time, in this third stage a new tool was introduced to solve this inconvenient increasing the efficiency in the deployment of the services.

3.3.1. Architecture

The core architecture of the web page remained the same (Node.js, Express, Angular and Bootstrap). In this third stage Redis database was introduced, the reasons for this selection are stated in section 2.3.4. The official Redis container from Docker Hub was used besides the Redis modules for Node.js.

At this stage of development and due to the big changes made in the design the module jsTree was removed. Instead the module for Node.js “fs” (File System) was used in order to read the content inside the folders for MPEG-DASH output and Premiere input created in the second version of the web app. As mentioned before, a web server, Nginx, was added in the architecture so that in case the web app crashed the media content could still be available.

Now that the architecture of the web app had become more complex a new method of deployment and configuration had to be introduced. The Docker Compose tool helped in that task. In a unique script written in YAML[13], various containers (Web app, Redis, Nginx and MPEG-DASH converter) could be configurated and deployed at the same time. This method organized the Docker architecture. Furthermore, in order to have a stable version of each container throughout the development, a stable version of each image was upload to Docker Hub, a Docker image repository. With Docker Hub and Docker Compose only one script was needed in order to run all the services, this reduced massively the deployment time.

The architecture of the DASH based distribution can be seen in Fig 3.10:

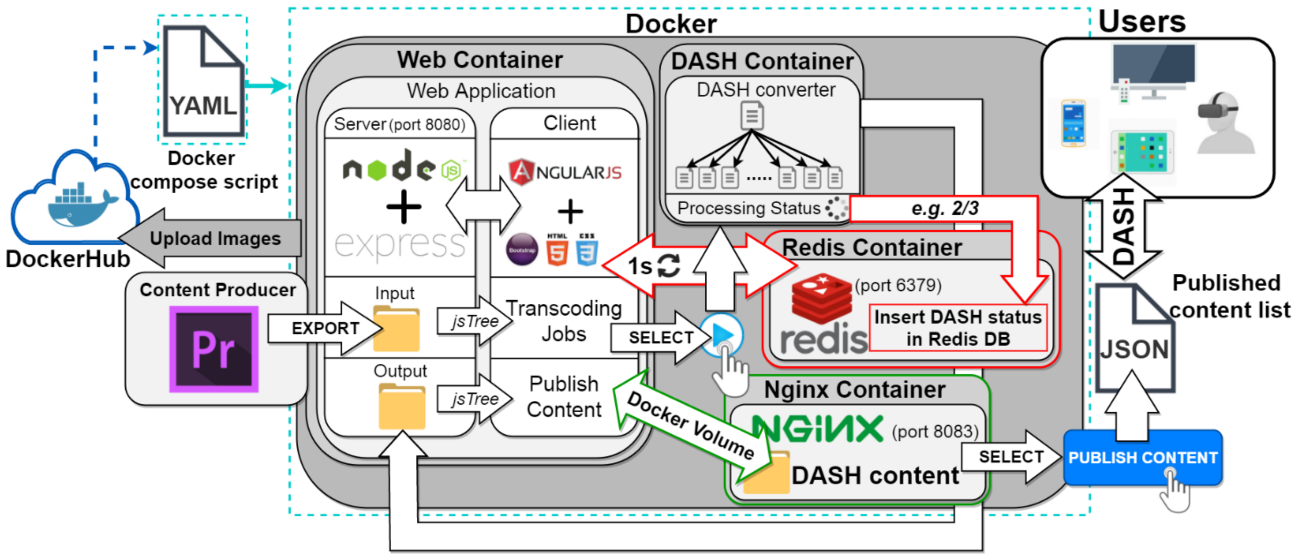


Fig. 3.10 DASH based distribution 2.0 web app architecture

[13] Human-readable data serialization language that takes concepts from other programming languages.

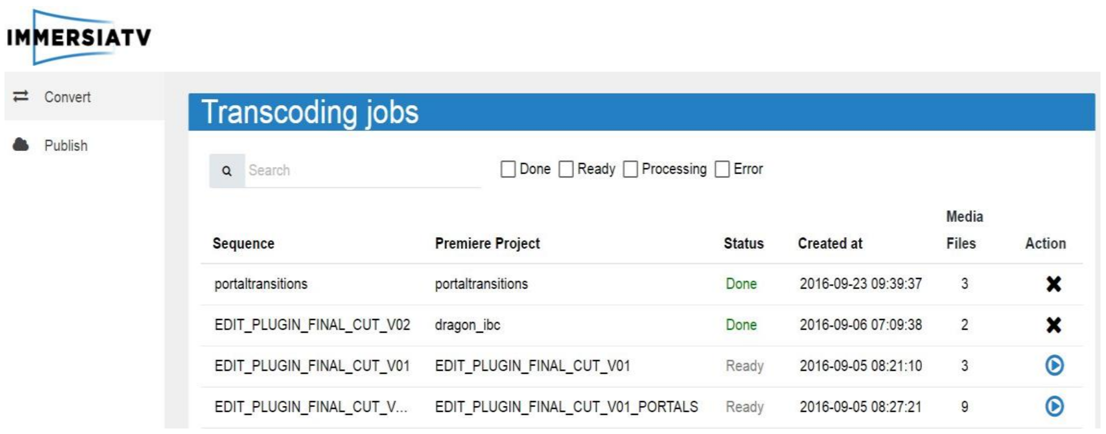
3.3.2. Problems encountered

As in all the development stages at the moment, some problems were encountered. The biggest problem was due to the technologies used, in order to update the MPEG-DASH processing state, every second a database request had to be made. This problem could get worst if the number of entries in the database increased increasing the loading times of the web app. Although in this third stage of development the possibility of converting simultaneous projects at the same time was possible, due to the fact the MPEG-DASH conversion used a big amount of CPU the maximum number of simultaneous conversions was three.

3.3.3. User experience

The view of the third web application development stage had two options in a left side menu: Convert and Publish.

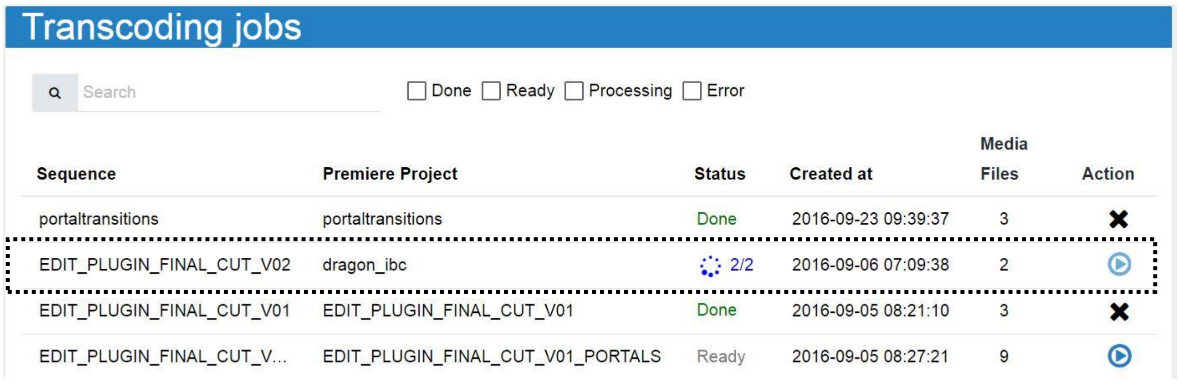
The first one is shown in Fig. 3.11. This view listed all the available projects that could be converted to MPEG-DASH format. Each element in the list had a Sequence (name given to the XML file), Premiere Project (name of the Premiere project), Status (could be between *Ready, Processing, Error* and *Done*), Created at (date of creation of the project), Media Files (number of media files inside the project) and Action (Convert or remove MPEG-DASH project). A searching option (only works with Sequence and Premiere Project) and status filtering option is situated over the available content table.



Sequence	Premiere Project	Status	Created at	Media Files	Action
portaltransitions	portaltransitions	Done	2016-09-23 09:39:37	3	✖
EDIT_PLUGIN_FINAL_CUT_V02	dragon_ibc	Done	2016-09-06 07:09:38	2	✖
EDIT_PLUGIN_FINAL_CUT_V01	EDIT_PLUGIN_FINAL_CUT_V01	Ready	2016-09-05 08:21:10	3	▶
EDIT_PLUGIN_FINAL_CUT_V...	EDIT_PLUGIN_FINAL_CUT_V01_PORTALS	Ready	2016-09-05 08:27:21	9	▶

Fig. 3.11 View of the DASH distribution v2.0 web app (Convert option)

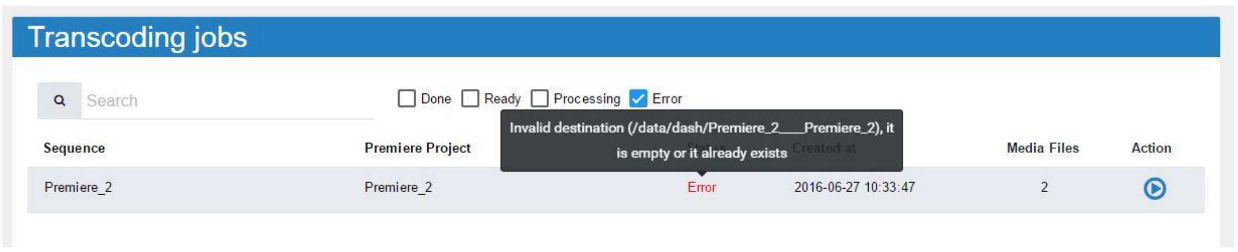
In this example we have four content projects in the list. There are two with status *Done*, this means the project has already been converted to MPEG-DASH thus the only action permitted is to remove the project. On the other hand there are two projects with status *Ready,* this means these projects have recently been exported thus the only action permitted is to convert these projects. By pressing the button bellow the “Action” column of one of the *Ready* projects, the status automatically changes to *Processing.* Actually “Processing” is what the database introduces, nevertheless in the web page, users view the MPEG-DASH processing status as shown in Fig. 3.12:



Sequence	Premiere Project	Status	Created at	Media Files	Action
portaltransitions	portaltransitions	Done	2016-09-23 09:39:37	3	✖
EDIT_PLUGIN_FINAL_CUT_V02	dragon_ibc	2/2	2016-09-06 07:09:38	2	▶
EDIT_PLUGIN_FINAL_CUT_V01	EDIT_PLUGIN_FINAL_CUT_V01	Done	2016-09-05 08:21:10	3	✖
EDIT_PLUGIN_FINAL_CUT_V...	EDIT_PLUGIN_FINAL_CUT_V01_PORTALS	Ready	2016-09-05 08:27:21	9	▶

Fig. 3.12 Processing status

If something goes wrong during the conversion the status changed to *Error*. If users hover over the word *Error,* the error will pop up as seen in Fig. 3.13:

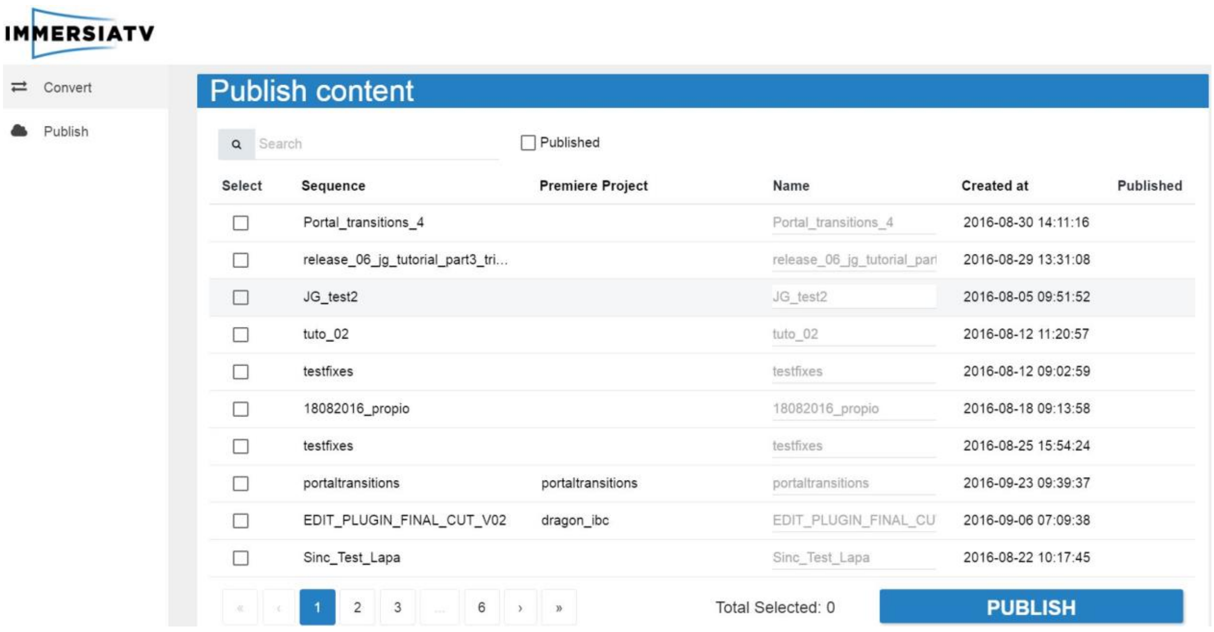


Invalid destination (/data/dash/Premiere_2___Premiere_2), it is empty or it already exists

Sequence	Premiere Project	Status	Created at	Media Files	Action
Premiere_2	Premiere_2	Error	2016-06-27 10:33:47	2	▶

Fig. 3.13 Error when converting to MPEG-DASH

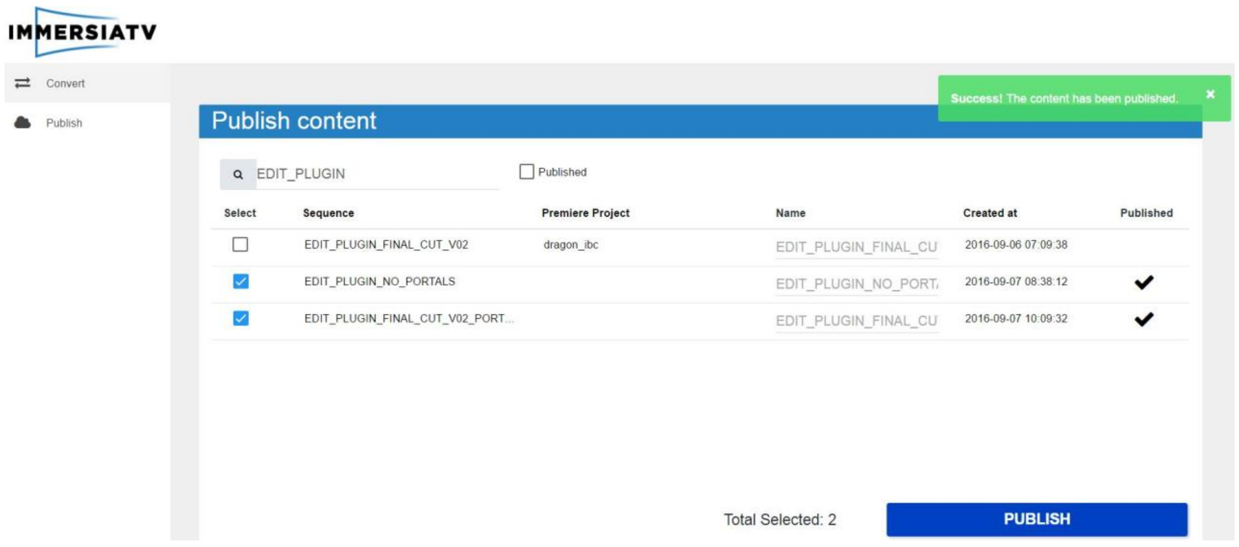
Fig. 3.14 shows the list content the users have available for publishing and viewing after with the Unity3D player.



Select	Sequence	Premiere Project	Name	Created at	Published
☐	Portal_transitions_4		Portal_transitions_4	2016-08-30 14:11:16	
☐	release_06_jg_tutorial_part3_tri...		release_06_jg_tutorial_pari	2016-08-29 13:31:08	
☐	JG_test2		JG_test2	2016-08-05 09:51:52	
☐	tuto_02		tuto_02	2016-08-12 11:20:57	
☐	testfixes		testfixes	2016-08-12 09:02:59	
☐	18082016_propio		18082016_propio	2016-08-18 09:13:58	
☐	testfixes		testfixes	2016-08-25 15:54:24	
☐	portaltransitions	portaltransitions	portaltransitions	2016-09-23 09:39:37	
☐	EDIT_PLUGIN_FINAL_CUT_V02	dragon_ibc	EDIT_PLUGIN_FINAL_CU	2016-09-06 07:09:38	
☐	Sinc_Test_Lapa		Sinc_Test_Lapa	2016-08-22 10:17:45	

Fig. 3.14 View of the MPEG-DASH distribution v2.0 web app (Publish option)

The table in the Publish view has three new columns: Select, Name and Published. In order to publish content, users have to select all the content they want to publish by checking the checkboxes in the Select column, then introduce in the Name column, the name they would like the Unity3D player to show as the title of the content (if the Name field is not filled, the Sequence name is taken as default name) and finally click the *PUBLISH* button. If everything went as expected, a checkmark has to appear in the Published column of each previous selected content as seen in Fig. 3.15:



Select	Sequence	Premiere Project	Name	Created at	Published
☐	EDIT_PLUGIN_FINAL_CUT_V02	dragon_ibc	EDIT_PLUGIN_FINAL_CU	2016-09-06 07:09:38	
☑	EDIT_PLUGIN_NO_PORTALS		EDIT_PLUGIN_NO_PORT/	2016-09-07 08:38:12	✓
☑	EDIT_PLUGIN_FINAL_CUT_V02_PORT...		EDIT_PLUGIN_FINAL_CU	2016-09-07 10:09:32	✓

Total Selected: 2

Fig. 3.15 Publishing content

As in the second stage, a contents JSON file is created with relevant information of each published content. This information will be read by the player in order to locate the content and make it available for the users (An example of this JSON file can be found in the Annex 2: Code).

3.4. Distribution based on Innovative formats

At this stage, the core of the web app had been developed. However a new functionality had to be developed. At the beginning of this document different VR content formats have been mentioned. Some of them are based in 3D models. The main purpose of this development stage is to include the possibility for users to publish this 3D content in order to consume it via a Unity3D player. This content will be converted previously to AssetBundles so the player can download and decompress it.

3.4.1. Architecture

The architecture of this development stage is practically the same with a difference, the 3D content has to be previously converted to an AssetBundle in order to compress all its components in a unique downloadable file. This transformation is done via Unity3D with the help of a script (The AssetBundle conversion script can be found in the Annex 2: Code).

The architecture of the MPEG-DASH-based and innovative distribution is seen in Fig. 3.16:

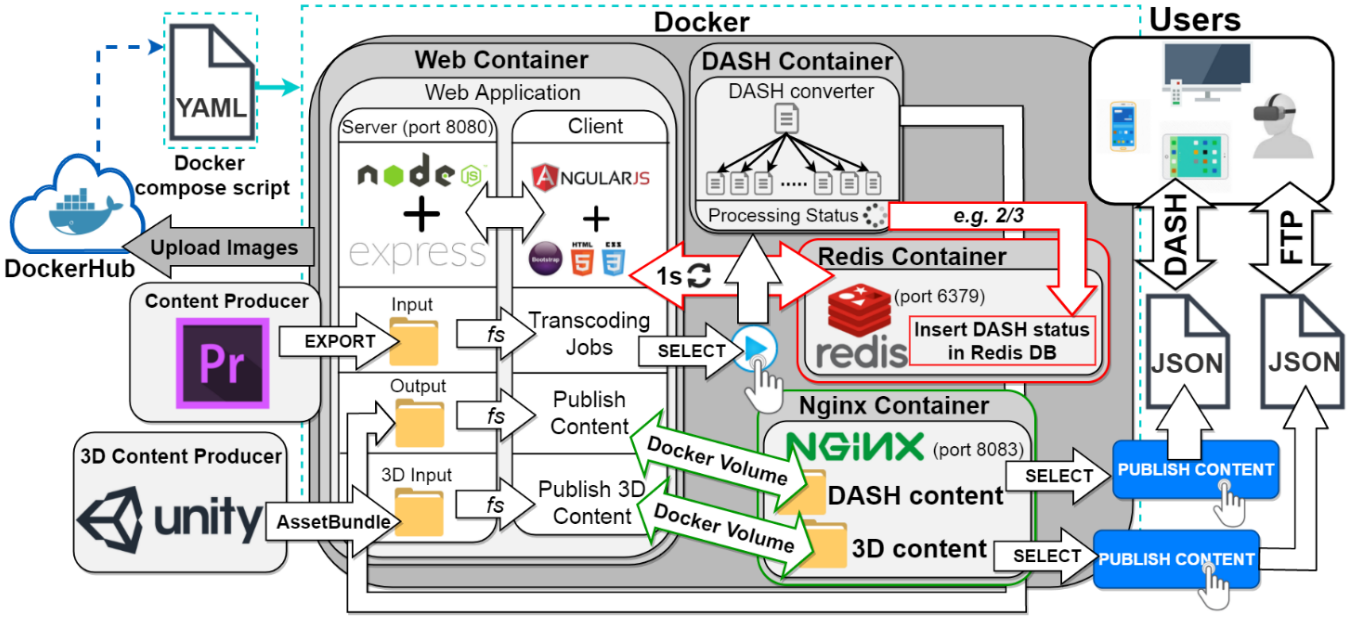


Fig. 3.16 MPEG-DASH-based and innovative formats distribution web app architecture

3.4.2. Problems encountered

The main problem during this stage was to find the suitable way to compress the 3D models into an AssetBundle. Traditional CGI along with sensor mesh based content were possible to compress as they are static elements, on the other hand, volumetric video content, as it is based on a group of 3D elements that create movement were not possible to compress into an AssetBundle for the moment. This kind of content had to be included entirely in the Unity player increasing the size of it as stated in section 1.2.4.

3.4.3. User experience

The user experience is the same as in the MPEG-DASH base distribution v2.0. The only difference is users have a new option to choose from the left hand side menu: 3D content. The view of this new option has the same appearance as the Publish view. Users can publish different innovative contents and view these after through a Unity3D player.

3.5. Conclusions of the development

As this web application was developed in order to distribute the content for ImmersiaTV, the development stages followed the progress of the project. Although ImmersiaTV is still under development, the web application satisfied what the project needed at that precise time, distribute VR content based in omnidirectional video through MPEG-DASH. Furthermore, the web application introduced the option to distribute sensor mesh-based 3D content.

In early development stages it what hard to believe the web application could work and look like it does. As complexity in the architecture increased, the deployment methods became harder until Docker was introduced. Docker has been a great discovery due to how it made easier the introduction of new features as a Redis database or an NGINX server without any headaches. It also helped, in terms of testing; how the web application worked perfectly regardless of OS, which was a big problem at the beginning of the development.

Throughout all the different stages many features didn't work as expected. The main one was users had to refresh the web page in order to view the new exported content. Another big issue was, once the Redis database was introduced, if changes in the data model had to be made it was difficult to deal with MPEG-DASH content added in the database within the old model. However as the Redis is a Key-Value NoSQL database old and new data models could coexist in the same database introducing new scripts.

Concluding, the goals of the web application have been achieved even though it has room for many improvements all of these stated in section 5.2.

CHAPTER 4. DEMO WITH A CONTENT PLAYER

This work had the main objective of developing a platform for innovative content distribution. As we mentioned earlier in the document, to test the omnidirectional video based content the ImmersiaTV player was used however for the sensor mesh-based content a different player was developed. Despite this work did not have the goal to develop this kind of applications, we decided to develop a simple app in order to visualize the difference between the three content formats: omnidirectional video, traditional CGI and sensor mesh-based. This application had an extra option where users can chose between been tracked by a CGI avatar or their own Kinect 2 live model.

The main purpose of this demo with the Unity player was to compare the PI and Psi between the three different environments with the two different options of body tracking. Due to lack of time, tests with real users could not be done. However a personal experience opinion will be given further in this document.

4.1. Environments: Omni, Traditional and Sensor based

As we mentioned before the demo is based in three different scenes each one based in an immersive content format. In order to have a consistent demo the three scenes were based on the same model. The model used was created from the coffee and water corner of i2cat offices in Barcelona. A picture of this corner can be seen in the Fig.4.1:

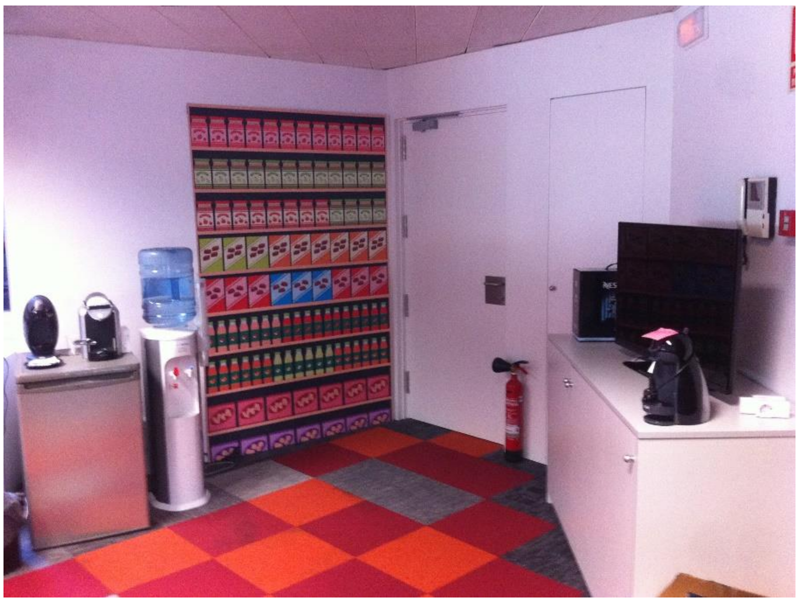


Fig. 4.1 Coffee and water corner at i2cat office

The reasons why this spot was chosen above other possibilities were it was the best suited environment inside the i2cat office (work development site) for the sensor based due to it needed internet connection in order to send the 3D information to the Skanect software. Furthermore, there was a traditional CGI model of the corner already modelled which removed the need of modelling one from scratch. However, due to hardware restrictions the spot was not the best one for omnidirectional video.

4.1.1 Omnidirectional video demo environment

Despite the demo had to be based in the same spot, as we mentioned earlier, the i2cat office coffee and water corner was not the best suited spot for omnidirectional videos. This was due to the omnidirectional cameras had to be placed at a minimum distance of 3m from everything. The 3m distance is needed to remove as many parallax effect as possible and to make the stitching possible as described in section 2.1.4.

Despite many attempts with different camera positioning, the result was not the satisfied one forcing to film another spot. The new chosen spot was the entrance to the Nexus I building at Campus Nord UPC in Barcelona, as seen in Fig. 4.2:

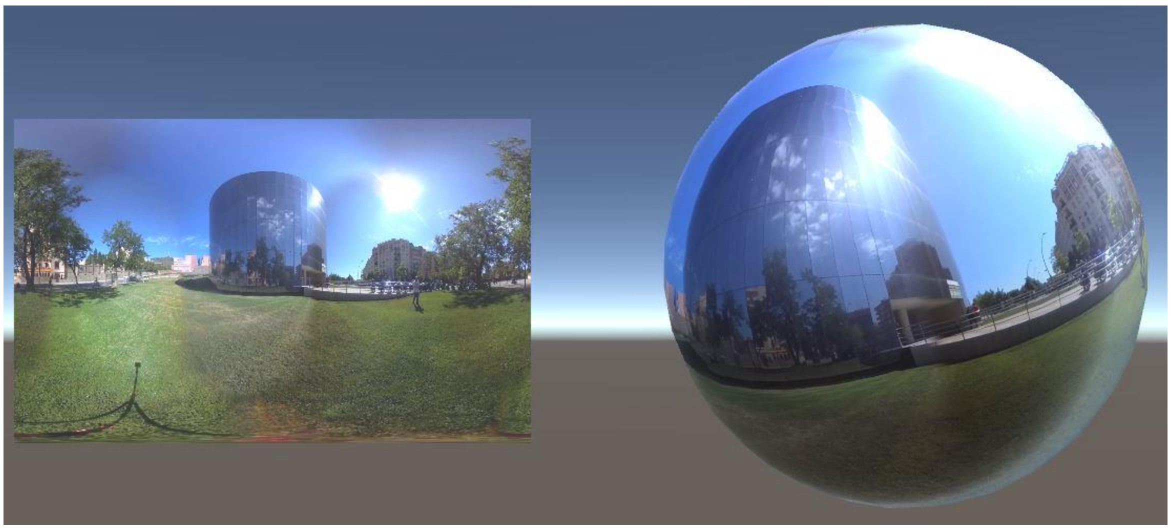


Fig. 4.2 Omnidirectional video environment (Nexus I building entrance)

Despite the difference in the location between the scenes, the experience is not affected and users can experiment the difference between the PI and Psi between the different content formats.

4.1.2 Traditional CGI demo environment

The traditional CGI model of the corner was created using Autodesk 3ds Max by a member of the i2cat staff. The colours of the different objects were changed in order to match the real ones. Being this a preliminary test the realism was not a priority.

As the speed of production was one of the key factors to measure, this demo did not need anything more realistic in order to view the difference in the user's experience between the three environments.

The traditional CGI model of the corner is seen in Fig. 4.3:

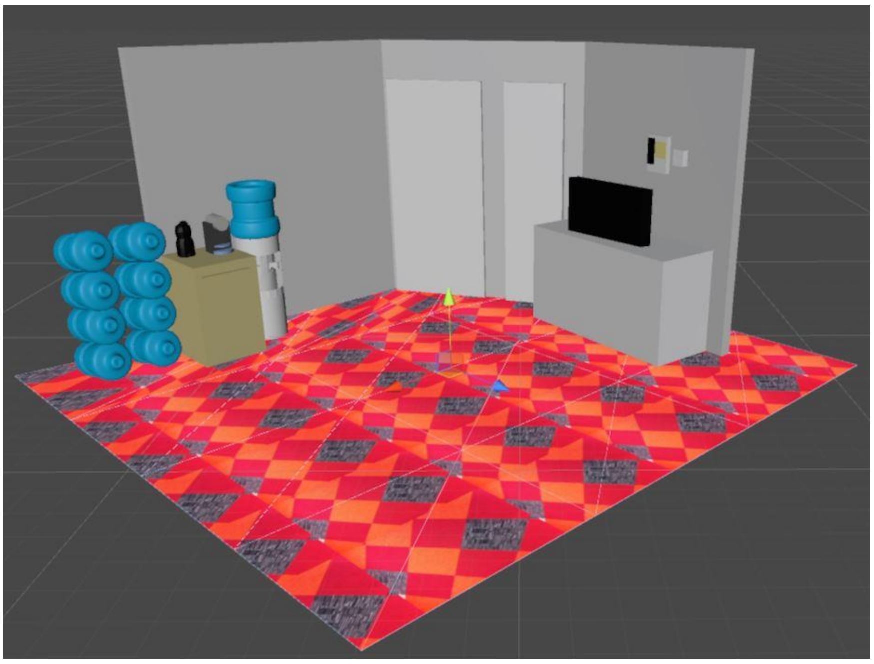


Fig. 4.3 Traditional CGI model of the coffee and water i2cat office corner

4.1.3. Sensor-based demo environment

The sensor based model of the i2cat corner was created using the Structure sensor together with the Skanect software. The Structure sensor is attached to an iPad and uses the Wi-Fi interface in order to send the 3D information, this caused several problems due to the fact the sensor could not connect with the software at first instance. The software uses a broadcast method in order to detect and connect with the sensor, this is why we had to create a local network with a router so both, software and sensor, could see each other.

The Structure sensor has several options regarding scanning options: objects and rooms. In this case the room scan option was used. This is the option used when trying to scan a closed area like a small room, however this area is recommended not to have detailed objects as the sensor's precision is not very accurate. For individual objects, the Structure sensor has other options.

The main problem of the sensor based model is the precision depends on the hardware. The model created for this demo was created in one unique scan, and no further editing was used. Although the quality of the model could be better, the speed of production compensates this drawback.

The sensor based model of the corner is seen in Fig. 4.4:

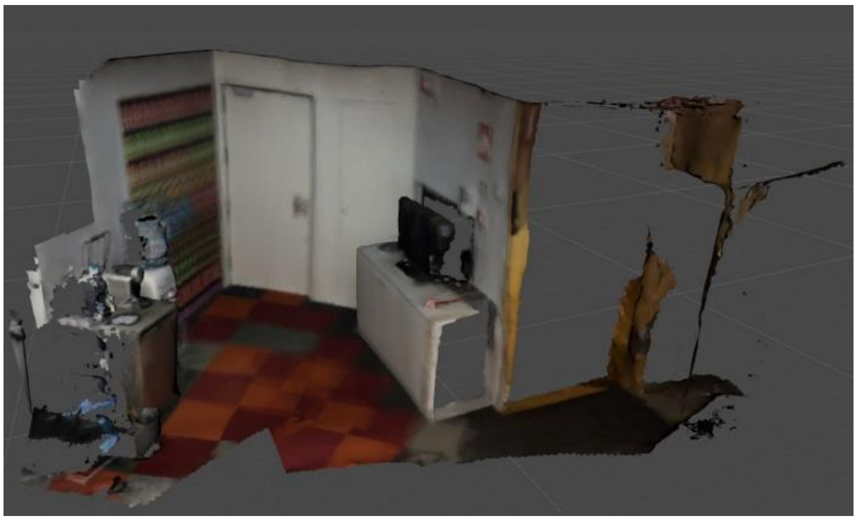


Fig. 4.4 Sensor based model of the coffee and water corner at i2cat office

4.2. User tracking options

In this demo users have the opportunity to choose between two different options for body tracking, traditional avatar or sensor based. Both options use the Kinect v2 SDK in order to track the user's skeleton. For users to see themselves inside the VR environment and somehow increase the PI and Psi, a mirror was added in the traditional CGI and sensor based scenes.

A user tracked within a traditional avatar (with a good configuration) had a better mobility inside the VR environment due to the avatar's joints are tracked with the Kinect's skeleton. However, if the user looks at the mirror or himself, the PI will decrease due to the user is not perfectly tracked with the avatar, also the movements have some delay and this is the key factor for the decrease in the PI.

On the other hand, a user tracked within a sensor based avatar using the Kinect v2 will have a better PI as the user will see himself tracked better inside the mesh. With this option the delay in movement was less comparing it with the CGI avatar. However, there were problems trying to track the sensor based model with the user which limited the movement. Furthermore as the Kinect only tracks what it has inside it FoV, the avatar was incomplete unlike the CGI example.

An example of both tracking options can be seen in Fig 4.5:

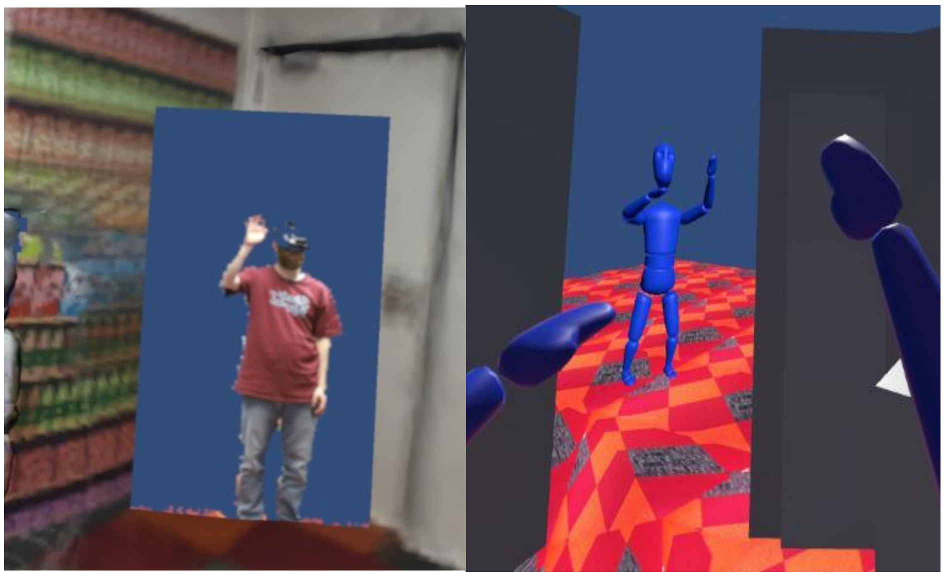


Fig. 4.5 Traditional CGI avatar and sensor-based avatar

4.3. Personal experience inside the demo testbed

The main difference between the three different environments was the movement limitations. The first time I was immersed inside the omnidirectional video scene I tried to move myself inside the scene with no success. Inside the omnidirectional video the perspective did not change when head movement, besides rotation, was made. On the other hand with the traditional CGI and sensor mesh-based scenes movement was not limited and objects became bigger as you got close to them, however the first reaction inside this scenes, especially in the traditional CGI, was to try and touch the 3D models in the scene; the result was like a ghost trying to catch something, as if the sense of touch had been disabled.

In terms of body tracking the best one was the Kinect 2 option as the delay between the movements was not as big as with the traditional CGI. Moreover, as you could see yourself through the mirror inside the VR experience, the quality of the experience was better.

In my opinion the best VR reality experience is inside the sensor mesh-based content or the traditional CGI, been tracked by the Kinect 2 sensor. However both scene options need more development time in order to create a decent experience for user testing.

CHAPTER 5. CONCLUSIONS AND FUTURE WORK

5.1. Achieved goals

This work had a main objective of developing a platform for distribution of innovative content. This objective has been achieved with the result of a functional web application that can distribute via MPEG-DASH content based in omnidirectional videos and via AssetBundles content based in 3D models.

On the other hand, although the work did not focus in the experience of the users but to distribute the content, a demo has been developed in order to test the functionalities of the web application. Moreover, this demo was developed creating three different VR environments representing three different content formats so users had the opportunity to experience the difference in PI and Psi between them. As mentioned before, due to lack of time no tests with users could be made despite this work was not aiming this.

The content shown in the demo due to no previous experience working with this kind of content, the results are far away from how it should have been in terms of 3D model quality. However the quality of the models does not affect drastically in the user experience.

Last but not least, volumetric video has been developed during some stages of the development of this work. This format has successfully been created in live conditions and also on demand as shown in the demo. However, unlike the other content formats, it was impossible to compress it in order to create an AssetBundle and publish it through the web application as stated in section 3.4.2.

5.2. Future work

In future work new functionalities for the web application can be developed. An option to view a preview visualization of the content could help users know more information about what are they going to publish. Furthermore along with this previsualization, the web application could give the user more information about the content as: the duration, audio and video codecs, resolution of the videos, etc.

Following the same lines, the database Redis could change to MongoDB [49] in order to create more complex database structure and increase the amount of information the web application offers about the media content. MongoDB is document-oriented unlike Redis that is based in key-value store, in other words, MongoDB is characterized by its schema-free organization of data. It supports more data types than Redis. The server-side scripts are written in JavaScript, Redis is written in Lua. As a last statement, along with Node.js, Express and Angular, MongoDB is part of the MEAN stack. In order to change the whole database, the Redis Docker will have to be change for a MongoDB Docker.

Moreover the npm module of Redis will have to be changed for the MongoDB module. However the biggest amount of time that will have to be invested will be in changing all the scripts in the web application. This task will not be trivial and a good organization will be needed in order to preserve the functionality of the web application.

At the moment the web application updates the status of the MPEG-DASH processing every second. A new technology named Socket.IO [50] could be introduced so the status only changes when changes are detected in the database, this would decrease the web application's loading time. Socket.IO is a JavaScript library for real-time web applications. It enable real-time, bi-directional communication between web clients and servers. Socket.IO is divided in two parts: a client-side library that runs in the browser, and a server-side library for Node.js. Socket.IO like Node.js is event-drive. In order to introduce Socket.IO in the web application, the npm module has to be downloaded. With these technology we can also solve the problem of having to refresh the web page for new content to be shown.

Other things related to the web application are the optimization of the process. A "hot folder"[14] functionality could be introduced in order to export from Premiere and automatically convert to MPEG-DASH without having to pass through the web page. Socket.IO could be used in order to detect changes in a folder, however adding the Chokidar [51] module we can achieve this. Chokidar is a Node.js module based in the fs module. Chokidar can listen for events inside a file directory like insert, delete or change.

In terms of the demo, many things could be improved. The first thing is to find a better spot in order to create the three environments. To create a better VR experience, the CGI and sensor based models have to be improved. At the moment the 3D models are based in a three wall environment where if the user looks further the only thing to see is a blue infinite horizon. In order to maximize the quality of the experience, the environment the user is immersed in has to be fully modelled or scanned, with a coherent background, not just blue everywhere. A good option would be mix the omnidirectional video content with the 3D models.

On the other hand, a compression format for the volumetric video created with the Kinect v2 has to be developed to decrease the amount of space consumed by the Unity application in order to play recorded volumetric video. This point is one of the most important ones in terms of distributing new innovative VR content as at the moment no volumetric video can be streamed through the internet. This new research area can change the way people communicate between each other through VR.

Last but not least, tests with real users would have to be done in order to have feedback about the PI and Psi of the different VR environments of the demo. This will help to choose which content format comparing speed of production versus quality of the experience is the best nowadays for content producers.

[14] A file system directory which is monitored by software so that any new files arriving in it can be processed.

5.3 Environmental impact

It is know that energy consumption has great impact in the environment. During the development of this work a server was used to deploy the web application and for MPEG-DASH conversion. As mentioned earlier in section 3.3.2 only three simultaneous conversions where possible which meant the server worked nearly with 100% of its CPU. Taking in consideration a working day of 8h, a server connected 24h (taking as an example [52]) can consume in average 0,437 kWh per day. According to the values presented in 2016 by the Ministry of Industry, Energy and Tourism of Spain [53], in reference to the data of 2012 the CO_2 emissions were 0,37 $kgCO_2/kWh$. This results in a total CO_2 emission of 162 grams of CO_2 per day.

In order to reduce the CO_2 emissions, more servers with less CPU consumption could be used in order to balance work. The MPEG-DASH conversion could be optimized in order to reduce the CPU consumption also. Servers that are not been used have to be turned off.

ACRONYMS

API	Application Program Interface
APP	Application
CGI	Computer-Generated Imagery
CPU	Central Processing Unit
CSS	Cascading Style Sheet
DSLR	Digital Single lens reflex
FoV	Field of View
FTP	File Transfer Protocol
GC	Google Cardboard glasses
HMD	Head Mounted Display
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
IP	Internet Protocol
JSON	JavaScript Object Notation
LAN	Local Area Network
LMZA	Lempel-Ziv-Markov chain Algorithm
MEMS	Microelectromechanical Systems
MPEG-DASH	Dynamic Adaptive Streaming over HTTP
NoSQL	Non Structured Query Language
OLED	Organic light-emitting diode
OS	Operating System
PI	Place illusion
Psi	Plausibility illusion
RAM	Random Access Memory
RTCP	Real-time Control Protocol
RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
RV	Realidad Virtual
SDK	Software Development Kit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Location
VR	Virtual Reality
XML	eXtensible Markup Language

Bibliography

[1] M. Slater, “Place illusion and plausibility can lead to realistic behaviour in immersive virtual environments,” *Philos. Trans. R. Soc. B Biol. Sci.*, vol. 364, no. 1535, pp. 3549–3557, Dec. 2009.

[2] The project: ImmersiaTV: http://www.immersiatv.eu/the-project-2/

[3] Horizon 2020 - European Commission: https://ec.europa.eu/programmes/horizon2020/.

[4] H. Schulzrinne, A. Rao, and R. Lanphier, *Real Time Streaming Protocol (RTSP)*. IETF, 1998.

[5] “Comparison of Streaming Formats.” http://bensoftware.com/blog/comparison-of-streaming-formats/

[6] General information about: Real Time Streaming Protocol, *Wikipedia, the free encyclopedia*. https://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol

[7] General information about: Dynamic Adaptive Streaming over HTTP, *Wikipedia, the free encyclopedia*. https://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP

[8] ExoPlayer Developr Guide: http://google.github.io/ExoPlayer/guide.html

[9] Why YouTube & Netflix use MPEG-DASH in HTML5: https://bitmovin.com/status-mpeg-dash-today-youtube-netflix-use-html5-beyond/

[10] Unity - Manual: AssetBundles: https://docs.unity3d.com/Manual/AssetBundlesIntro.html

[11] Assets, Objects and serialization,” *Unity*. https://unity3d.com/ru/learn/tutorials/topics/best-practices/assets-objects-and-serialization

[12] Unity - Manual: Asset Bundle Compression: https://docs.unity3d.com/Manual/AssetBundleCompression.html

[13] Information about 7z Format: http://www.7-zip.org/7z.html

[14] General information about: Lempel–Ziv–Markov chain algorithm, *Wikipedia, the free encyclopedia*: https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Markov_chain_algorithm

[15] General information about: LZ4 (compression algorithm), *Wikipedia, the free encyclopedia*: https://en.wikipedia.org/wiki/LZ4_(compression_algorithm)

[16] AssetBundles and the AssetBundle Manager, *Unity*: https://unity3d.com/es/learn/tutorials/topics/scripting/assetbundles-and-assetbundle-manager

[17] Studio user guide| VideoStitch, *360 VR Video software* | *VideoStitch*: http://www.video-stitch.com/documentation/user-guide/

[18] Riley from Call of Duty Ghosts 2013: http://www.cgrecord.net/2016/01/riley-from-call-of-duty-ghosts-2013.html

[19] General information about: HTC Vive," *Wikipedia, the free encyclopedia*: https://en.wikipedia.org/wiki/HTC_Vive

[20] 3D scanning software Skenct 3D: http://skanect.occipital.com/

[21] Structure sensor information: http://www.3ders.org/articles/20130917-capture-the-world-in-3d-structure-sensor-turns-ipad-into-a-3d-scanner.html

[22] E. Rodgers, "DSLR and Kinect combine to produce dream-like visuals," *The Verge*, 02-May-2012: http://www.theverge.com/2012/5/2/2993117/3d-filmmaking-open-source-RGBD

[23] DepthKit website: http://www.depthkit.tv/

[24] Kinect v2 Examples with MS-SDK - Asset Store: https://www.assetstore.unity3d.com/en/#!/content/18708

[25] B. Kuchera, "The complete guide to virtual reality in 2016 (so far) (Update: February 2016)," *Polygon*, 15-Jan-2016: http://www.polygon.com/2016/1/15/10772026/virtual-reality-guide-oculus-google-cardboard-gear-vr

[26] General information about: Google Cardboard, *Wikipedia, the free encyclopedia*: https://en.wikipedia.org/wiki/Google_Cardboard

[27] Google Cardboard - Visores oficiales de realidad virtual - Google Store: https://store.google.com/product/google_cardboard

[28] B. Lang, "Samsung Gear VR Specifications Reveal Sub-20ms Latency," *Road to VR*, 03-Sep-2014: http://www.roadtovr.com/samsung-gear-vr-official-specifications-20ms-latency/

[29] Gafas Gear VR para smartphones Galaxy | SAMSUNG, *Samsung ES*: http://fb.es.samsung.com/consumer/mobile-devices/wearables/gear/SM-R322NZWAPHE

[30] Image of HTC Vive: http://imagenes.lifeinformatica.com/HTC-99HAHZ046-00/imgs/99HAHZ046-00.png

[31] QBiC-MS-1-Instruction-Manual.pdf: http://elmousa.com/wp-content/uploads/2016/03/QBiC-MS-1-Instruction-Manual.pdf

[32] QBiC PANORAMA X, *Elmo USA*: https://www.elmousa.com/product/qbic-panorama-x/

[33] Parallax diagram description foto from Dr. Daniel Lau: http://lau.engineering.uky.edu/author/dlau/

[34] Internship at Inria: Parallax compensation on 360° panoramic video: http://devernay.free.fr/vision/jobs/2013panoramic.html.

[35] Structure Sensor Support Center | What are the Structure Sensor's technical specifications?: http://structure.io/support/what-are-the-structure-sensors-technical-specifications

[36] Structure Sensor - 3D scanning, augmented reality, and more for mobile devices: http://structure.io/

[37] Kinect hardware information: https://developer.microsoft.com/en-us/windows/kinect/hardware

[38] Kinect image from Google images: https://upload.wikimedia.org/wikipedia/commons/f/f6/Xbox-One-Kinect.jpg

[39] General information in Docker (software), *Wikipedia, the free encyclopedia:* https://en.wikipedia.org/wiki/Docker_(software)

[40] General information in GStreamer, *Wikipedia, the free encyclopedia:* https://en.wikipedia.org/wiki/GStreamer

[41] General information in Node.js, *Wikipedia, the free encyclopedia:* https://en.wikipedia.org/wiki/Node.js

[42] General information in Express.js, *Wikipedia, the free encyclopedia:* https://en.wikipedia.org/wiki/Express.js

[43] General information in AngularJS, *Wikipedia, the free encyclopedia:* https://en.wikipedia.org/wiki/AngularJS

[44] General information in Bootstrap, *Wikipedia, the free encyclopedia:* https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)

[45] Redis official page: http://redis.io/

[46] General information about 360-degree video, *Wikipedia, the free encyclopedia*: https://en.wikipedia.org/wiki/360-degree_video

[47] Immersive Video for Virtual Reality | Vrideo: http://www.vrideo.com/

[48] VRapp.co | Experience and Share Virtual Reality on the Web: https://vrapp.co/

[49] "MongoDB vs. Redis Comparison: http://db-engines.com/en/system/MongoDB%3BRedis

[50] General information about Socket.IO, *Wikipedia, the free encyclopedia:* https://en.wikipedia.org/wiki/Socket.IO

[51] Chokidar module GitHub: https://github.com/paulmillr/chokidar.

[52] A. J. Pablo, ¿Cuánta energía gasta un ordenador? (aproximaciones): http://www.leantricity.es/cuanta-energia-gasta-un-ordenador-aproximaciones/

[53] "Valores de emisiones publicados en otros documentos" Ch. 6 sec. 6.1 de Factores de emisión de CO2 y coeficientes de paso a energía primaria de diferentes fuentes de energía final consumidas en el sector de edificios en España: http://www.minetur.gob.es/energia/desarrollo/EficienciaEnergetica/RITE/Reconocidos/Reconocidos/Otros%20documentos/Factores_emision_CO2.pdf

Annex 1: ImmersiaTV Project

ImmersiaTV: broadcasting multi-platform experiences [1]

Joan Llobera, Juan Núñez, Sergi Fernández Langa
i2cat Foundation

ABSTRACT

In the contemporary living room, the audience's attention is often divided between TVs, second screens and, increasingly, head mounted displays. To address this reality, ImmersiaTV is a H2020 European project which is redefining the end-to-end broadcast chain: production, distribution and delivery. It is built on two ideas: multi-platform synchronous content delivery, and orchestrated videos rendered in the head-mounted display as interactive inserts, which allow introducing basic interactive storytelling techniques (scene selection, forking paths, etc.) as well as classical audiovisual language that is not possible to render with 360 videos (close-ups, slow motion, shot-countershot, etc). We here report on our early efforts for multi-platform synchronous delivery as well as video orchestration to introduce interactive inserts.

Keywords
Video synchronization, television (TV), head-mounted display (HMD), omnidirectional video (ODV)

1. INTRODUCTION

The majority of European TV consumers now watch TV programs in a multi-display environment. Second screens - mostly smartphones, tablets or laptops- are generally used to check information not directly related to the events in the TV content being watched. As a result, the attention of the audience is generally divided between these different streams of information. Broadcasters have tried to orchestrate all these different rendering platforms to complement each other consistently. However, their success is limited, and this limited success is due, at least in part, to the very different formats in which information is delivered (web-based texts, mobile apps, traditional broadcast television…).

ImmersiaTV is delivering a new form of broadcast omnidirectional video that offers end-users a coherent audiovisual experience across head mounted displays, second screens and the traditional TV set, instead of having their attention divided across them. This new experience seamlessly integrates with and further augments traditional TV and second screen consumer habits. In other terms: the audience is still be able to watch TV sitting on their couch, or tweet comments about it. However, by putting omnidirectional content at the center of the creation, production and distribution processes, the audience is also able to use immersive displays to feel like being inside the audiovisual stream. ImmersiaTV is built on two ideas: multi-platform synchronous content delivery, and orchestrated videos rendered in the head-mounted display as interactive inserts, which allow, first, introducing basic interactive storytelling techniques such as scene selection, forking paths, etc. and, second, reintroducing classical audiovisual language that is not possible to render with 360 videos, such as close-ups, slow motion, shot-countershot, etc. (see also Fig. 1).

Designing and implementing a broadcast audiovisual production chain is a challenging due to the diversity of processes, technologies and production practices that it requires. In the remainder of this document we outline the main solutions, either adopted or implemented, to demonstrate the practical feasibility of creating and delivering content adapted to this approach. For this purpose we introduce examples from the first pilot of ImmersiaTV, focused on a documentary showing the day life of a kid attending the Porto Football School.

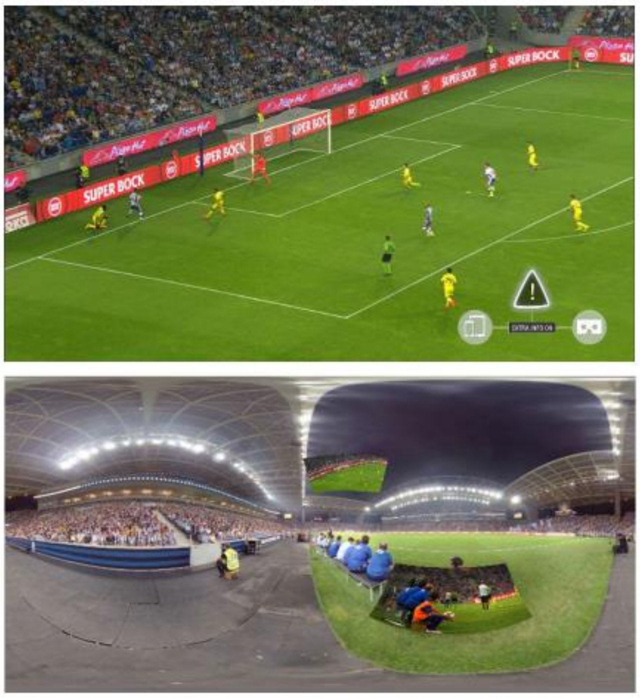


Figure 1: Top: an image typical of a traditional TV showing a football match. An insert informs the consumer that content is also available for tablets and VR headsets. Bottom: a capture of an omnidirectional video with inserts of traditional cameras. This content is delivered synchronized with the main TV stream. Image courtesy of Lightbox (www.lightbox.pt).

[1] Submitted at the NEM Summit 2016 http://nem-initiative.org/ This work has been funded by European Union's Horizon 2020 program under grant agreement nº688619.

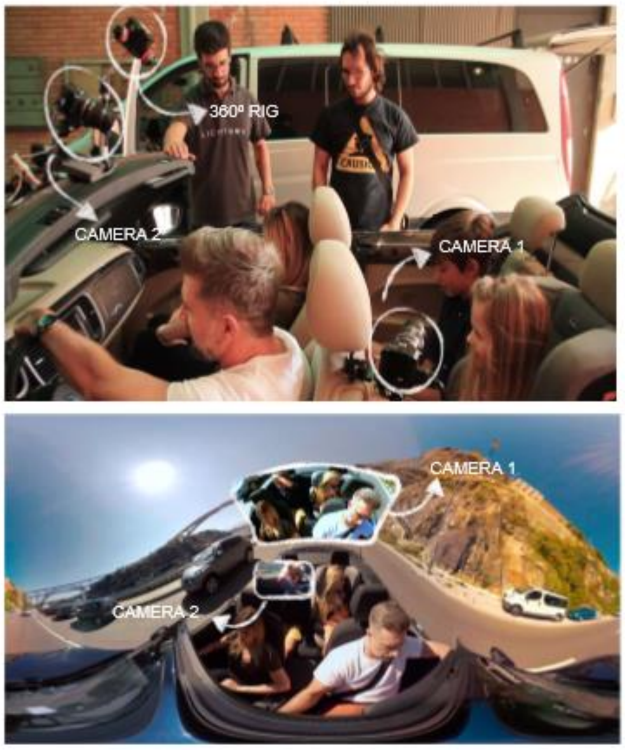


Figure 3: Top: a camera setup to record traditional and omnidirectional video simultaneously. Bottom: a schematic diagram of possible directive inserts located within the omnidirectional video. Image courtesy of Lightbox (www.lightbox.pt).

2. CAPTURE

The creation of content that is both omnidirectional and traditional requires shooting simultaneously in both content formats. Preliminary tests with separate shootings for omnidirectional and traditional cameras revealed it was unfeasible to synchronize two different shootings, even when the actors in the scene were repeating the same actions.

The solution found by the production team was to use two *BlackMagic Micro Studio Camera 4k* micro-cameras for the traditional shooting, which could be hidden or, if visible, removed in post-production with a reasonably small amount of effort. This combined with an omnidirectional capture rig composed of 6 GoPro 3 Black Rig cameras allowed capturing simultaneously traditional and omnidirectional footage. However, for a joint shooting, we must address the fact that omnidirectional cameras capture the whole visual field, and therefore would show the traditional camera and the film crew behind it. This is not problematic for sports or music events, but it goes strongly against the conventions of fiction or documentary.

3. EDITION

Dedicated stitching tools such as video-stitch studio by Video-stitch, or Autopano by Kolor, allow stitching video footage captured with camera rigs in order to create omnidirectional videos such as showed the examples shown in figure 1 and figure 2. However, editing content targeting synchronous rendering across devices is not possible with currently available edition tools.

To address this fact, we have designed and implemented a plugin for Adobe Premiere. The ImmersiaTV plugin shown in figure 3 allows defining the inserts that are placed within an omnidirectional video, as well as which tracks should be rendered in each of 3 possible devices (TV, tablet or VR

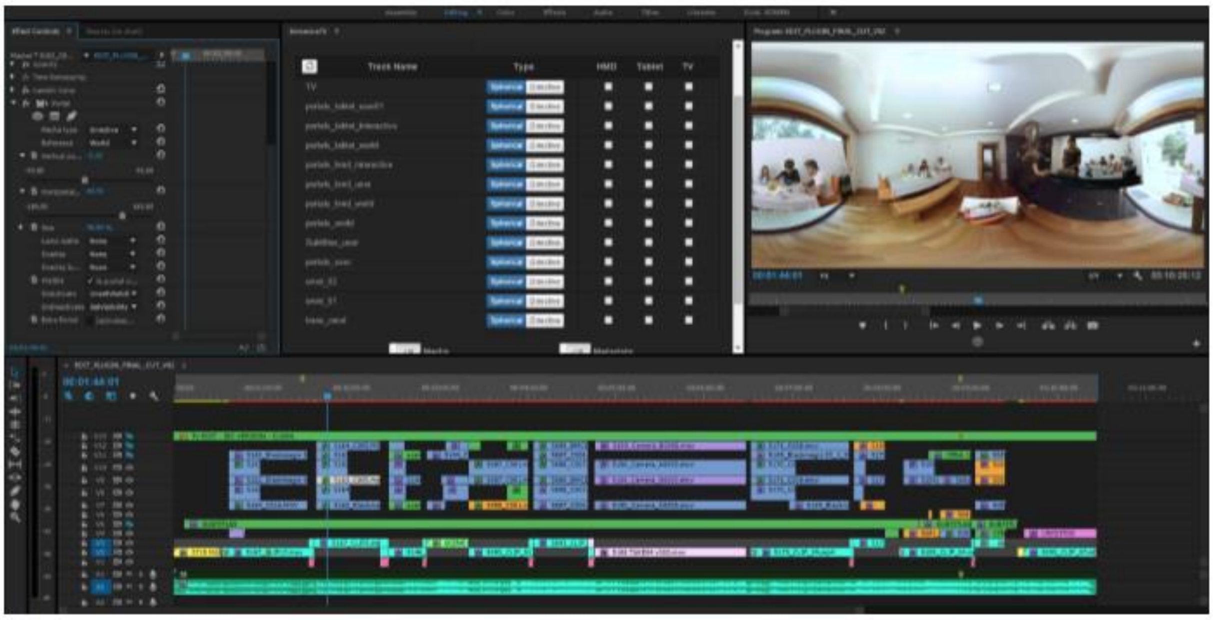


Figure 2: The Adobe Premiere ImmersiaTV panel, shown at the center, allows defining omnidirectiona l l and directive (i.e., traditional) tracks, as well as which devices does each track target. The inserts added to the omnidirectional view, shown at right, can be edited with the ImmersiaTV Portal Effect, whose widgets are shown at the left. Image courtesy of Lightbox (www.lightbox.pt).

googgles). It works both with Mac and Windows, and we have validated that, after going through a tutorial, can use it to create multi-platform content. Further work should refine and expand the possibilities to introduce interactive capabilities, in order the user's input affects the media being rendered or other aspects of the experience.

4. DISTRIBUTION

The media encoding uses readily available and methods, concretely H.264 and AAC encoding, and adaptative bitrate streaming based on MPEG-DASH (ISO/IEC 23009-1:2014). Encoding is implemented as a cloud service, running on a Linux server using the Dockers virtualization tool as well as MP4Box from Gpac's MP4Box for MPEG-DASH multiresolution encoding[2].Video decoding uses the Gstreamer library[3].

The additional metadata required for playout, which relates audiovisual streams with devices (i.e., allows selecting different streams for TVs and tablets), as well as to define interaction and media orchestration requirements, follows closely the format of MPEG-DASH manifests, and its XML specification is publicly available[4]

Content publication is performed through a custom built website which allows triggering media conversion, as well as monitoring progress on media encoding and publishing content generating a list of content parsed at the delivery stage. This solution is appropriate for offline production. Live production requires a complete different set of tools, for which we are currently in the early phases of architecture design and requirement specification.

5. DELIVERY

Metadata parsing is done with a custom parser, which also generates the appropriate geometry and provides the DASH player with the appropriate DASH manifests.

Synchronized multi-platform delivery is still a challenging task. Currently we have implemented proofs-of-concept of synchronized delivery using 2 synchronization protocols:

1. Synchronization based on emerging broadcast standards (see reference DTS/JTC-DVB-343, from the European Telecommunications Standard Institute)
2. Synchronization based on Gstreamer's version of the Precision Time Protocol (IEEE 1588)

Delivery of omnidirectional video requires integrating the user's input with the experience, in order to select the right portion of the omnidirectional view. We are currently exploring two different software architectures for media playout. A first version is based on a custom integration of Gstreamer and the Unity3D game engine which we have released opensource.[5].

A second version is based entirely on web technologies, and content is rendered within a browser. We are using custom code combined with three.js[6] for the geometry generation and shaka-player[7] for DASH rendering.

We have not yet completed systematic evaluation of these implementations, initial benchmarking tests shows both players, when executed on a Samsung s6 with android 6.0 can play one 4K video. However, the additional processing power required for metadata parsing imposes reductions on the playout quality. In addition, synchronization across devices necessarily imposes seeking through a video stream being consumed, which becomes slower at higher rates. Rendering for mobile-based head mounted displays, either cardboard or Samsung GearVR, also imposes a double rendering process (one for each eye), which further decreases performance. Therefore, in practice we can currently reproduce synchronously video up to 4096x2048, bitrate of 50 Mega bits per second (Mbps) and 30 frames per second on a Samsung S6, but this resolution drops to 1024x512, bitrate of 4Mbps and 25 frames per seconds when VR rendering is required.

The browser based rendering, in practice, can reproduce up to 2048x512, with 12Mbps of bitrate and 30frames per second.

It seems clear that the limits in the quality that we can deliver is determined by hardware processing load, rather than bandwidth limitations. Further work will, first, develop more precise performance tests and, second, implement strategies to improve the data throughput that can be processed with off-the-shelf hardware, particularly when different media streams need to be synchronized in one device and where the orchestration of these media streams is affected by the consumer's input.

6. CONCLUSIONS

Our recent efforts to assemble an innovative end-to-end production and distribution pipeline show that this endeavor is a challenging task which requires combining several software technologies. However, the availability of open source tools as well as the existence of APIs to develop plugins for proprietary editing software makes possible, today, to design, implement and demonstrate an innovative end-to-end broadcast pipeline from scratch in a period of 8 months. For our case, it required, approximatively, 8 software engineers with different profiles, and a mean dedication of four months each.

Further work will be needed to optimize the media quality delivered, as well as to develop examples of content exploring more exhaustively the possibilities in terms of interaction that such a platform offers.

Adapting this paradigm to live production will also require implementing or adapting a different tool for live editing, as well as integrating a live MPEG-DASH encoding service.

[2] https://gpac.wp.mines-telecom.fr/mp4box/

[3] https://gstreamer.freedesktop.org/

[4] http://server.immersiatv.eu/public_http/metadata/ImmersiaTV.html

[5] https://www.assetstore.unity3d.com/en/#!/content/59897

[6] https://threejs.org/

[7] https://github.com/google/shaka-player

Annex 2: Code

2.1. Dockerfile example

```
FROM ubuntu:16.04
    MAINTAINER Einar Meyerson <einar.meyerson@i2cat.net>
RUN apt-get update && \
    apt-get install -y --no-install-recommends software-properties-
common
RUN apt-get update && apt-get install -y \
    python \
    git \
    make \
    npm \
    curl \
    nodejs

RUN ln -s /usr/bin/nodejs /usr/bin/node
RUN npm install -g bower

COPY Content_JSONs /var/www/api/Content_JSONs
COPY package.json /var/www/api/
COPY README.md /var/www/api/
COPY routes /var/www/api/routes
COPY server.js /var/www/api/
COPY www /var/www/api/www

WORKDIR /var/www/api
RUN npm install && npm install forever -g
RUN cd www && bower install --allow-root && cd ..

COPY run.sh /usr/local/bin

RUN apt-get install -y apt-transport-https ca-certificates
RUN apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --
recv-keys 58118E89F3A912897C070ADBF76221572C52609D
RUN echo "deb https://apt.dockerproject.org/repo ubuntu-xenial main"
> /etc/apt/sources.list.d/docker.list
RUN apt-get update
RUN apt-get install -y --no-install-recommends docker-engine

RUN chmod +x /usr/local/bin/run.sh

#RUN groupadd docker && usermod -a -G docker `whoami`
EXPOSE 8080

CMD ["run.sh"]
```

2.2. Script run.sh

```
#!/bin/bash

export PREMIEREPATH="/data/premiere"
export HOTPATH="/data/hot_dash"
export DASHPATH="/data/dash"
export HOST_IP

forever stopall
forever start server.js
forever --fifo logs 0
```

2.3. Metadata XML example

```
<ITVEvents xmlns="urn:immersiatv:immersiatv01:2016:xml" xmlns:xsi="h
ttp://www.w3.org/2001/XMLSChema-
instance" xsi:schemaLocation="urn:immmersiatv:immersiatv01:2016:xml
http://server.immmersiatv.eu/public_http/metadata/ImmersiaTV.xsd" ty
pe="static">
<DefineScene id="0" device="hmd" time="0">
<DefineShape id="1" mediaFile="sp_trans_omni_0_1_2_5/sp_trans_omni_0
_1_2_5" type="sphericalCap">
<Anchor id="0" distance="1.00"/>
</DefineShape>
</DefineScene>
<DefineScene id="1" device="tablet" time="0">
<DefineShape id="1" mediaFile="sp_trans_omni_0_1_2_5/sp_trans_omni_0
_1_2_5" type="sphericalCap">
<Anchor id="0" distance="1.00"/>
</DefineShape>
</DefineScene>
<DefineScene id="2" device="tv" time="0">
<DefineShape id="1" mediaFile="tv_tv_12/tv_tv_12" type="rectangle"><
/DefineShape>
</DefineScene>
<DefineScene id="3" device="hmd" time="626.440002441406">
<RemoveShape id="1"/>
</DefineScene>
<DefineScene id="4" device="tablet" time="626.440002441406">
<RemoveShape id="1"/>
</DefineScene>
<DefineScene id="3" device="tv" time="626.440002441406">
<RemoveShape id="1"/>
</DefineScene>
</ITVEvents>
```

2.4. JSON content file example

```
{
title: "ImmersiaTV content server",
content:
[
        {
                name: "Portal_transitions_4",
                hash: "2c77c1c4c6400938e6d89f2d7bfa33ed",
                url: "http://192.168.10.115:8083/dash/Portal_transitions_4___PortalTransitionEntradaSalida/Portal_transitions_4.xml"
        },
        {
                name: "release_06_jg_tutorial_part3_trigger_insert_appear",
                hash: "e2072fb422e36a2a69b3ddb2fd77e165",
                url: "http://192.168.10.115:8083/dash/release_06_jg_tutorial_part3_trigger_insert_appear___Release_06_trigger/release_06_jg_tutorial_part3_trigger_insert_appear.xml"
        },
        {
                name: "18082016_propio",
                hash: "b5f380caf837eb00f6e9be44c02fda00",
                url: "http://192.168.10.115:8083/dash/18082016_propio/18082016_propio.xml"
        },
        {
                name: "Sinc_Test_Lapa",
                hash: "2a4b59045ba19c2cccd627705e39d830",
                url: "http://192.168.10.115:8083/dash/Sinc_Test_Lapa/Sinc_Test_Lapa.xml"
        }
        ]
}
```

2.5. AssetBundle script

```
using UnityEditor;

public class CreateAssetBundles
{
    [MenuItem ("Assets/Build AssetBundles")]
    static void BuildAllAssetBundles ()
    {
        BuildPipeline.BuildAssetBundles ("Assets/AssetBundles",
BuildAssetBundleOptions.None, BuildTarget.StandaloneOSXUniversal);
    }
}
```