Deliverable

Deliverable name
D3.3 – Report on Immersive Media Player Integration

Project Acronym:	IMMERSIFY
Grant Agreement number:	762079
Project Title:	Audiovisual Technologies for Next Generation Immersive Media

Revision:	1.0
Authors:	Diego Felix de Souza (SD), Ali Nikrang (AEF), Clemens Scharfen (AEF), Ivor Diosi (AEF), Mikołaj Węgrzynowski (PSNC), Erik Sundén (NVAB)
Delivery date:	M33
Dissemination level (Public / Confidential)	Confidential
Reviewers	Szymon Malewski (PSNC)

Abstract

This report details the progress made in WP3, in particular, Task 3.5 - Media player integration in combination with the following project tasks: Task 3.1 - Decoder and renderer performance optimization, Task 3.2 - Decoder with VR specific features, Task 3.3 - Multi-screen and scalable display, and Task 3.4 - Immersive Audio. The features developed under those tasks were combined in an SDK to facilitate the creation of immersive media players. In the Immersify project, the following players have been developed and reported herein:

- Spin Digital SpinPlayer and streamplay
- Ars Electronica Immersify Player for Deep Space 8K
- Ars Electronica Unity3D Immersify Player
- Ars Electronica Unreal Engine 4 Immersify Player
- PSNC 8K 3D Media Player
- PSNC Cave Media Player
- NVAB Dome Player

For each of the reported players, this deliverable describes the work performed, results achieved, progress made beyond the state-of-the-art, and deviations concerning the original work plan.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement 762079.

REVISION HISTORY

Revision	Date	Authors (Entity)	Description of changes
0.1	05.06.2020	SD, AEF, PSNC and NVAB	Version ready for review
0.2	08.06.2020	Szymon Malewski (PSNC)	Review
1.0	19.06.2020	Diego Felix de Souza	Format revision

CONTRIBUTORS

Contributor's name	Entity	Contact e-mail
Diego Felix de Souza	SD	diego@spin-digital.com
Mauricio Alvarez Mesa	SD	mauricio@spin-digital.com
Sergio Sanz-Rodriguez	SD	sergio@spin-digital.com
Thomas Siedel	SD	thomas@spin-digital.com
Clemens Scharfen	AEF	clemenssc@aec.at
Ivor Diosi	AEF	ivordi@aec.at
Ali Nikrang	AEF	alini@aec.at
Roland Haring	AEF	rolandha@aec.at
Mikołaj Węgrzynowski	PSNC	mwegrzyn@man.poznan.pl
Szymon Malewski	PSNC	szymonm@man.poznan.pl
Maciej Jaśkiewicz	PSNC	mjaskiewicz@man.poznan.pl
Erik Sundén	NVAB	erik.sunden@liu.se

TABLE OF CONTENTS

Introduction **6**

Task 3.5 Description **7**

Original Task Description 7

Deviations from the Description of Work 7

Relevance of the Task to the Project 8

New Immersive Features **9**

Introduction 9

Equirectangular Projection (ERP) 9

Curved Screen Support with Cylindrical Projection 9

Multi-device Support with Interactive 360° Video 10

Messages 12

Callback function 12

360° Navigation with the Xbox Controller 12

New Licensing Mode: needed for the release of 3D engines plugins 14

HOA Sound Extension 15

Custom Loudspeaker Layouts 16

Rotation and Zoom for up to 7th Order 17

Spin Digital SDK and Media Players **19**

Concept 19

Implementation 20

Integration Phase 1: “16K video playback” 20

Integration Phase 2: “Multi-device and responsive 360° playback” 21

Integration Phase 3: “Immersive and interactive 360° video with spatial audio” 23

Results 26

Specification of media playback systems 29

Integration Phase 1: 16K video playback 30

Integration Phase 2: Multi-device and responsive 360° playback 30

Integration Phase 3: Immersive and interactive 360° video with spatial audio 31

Advances over the State of the Art 31

Native Core Plugin and Immersify Video Player **34**

Concept 34

Implementation 35
Native Core Plugin 35
Immersify Video Player 36
Results 39
Specification of media playback systems 40
Advances over the State of the Art 40
Unity Immersify Player **41**
Concept 41
Core Components 42
Shaders 42
Scenes 42
Implementation 43
Using the Unity Immersify Video Plugin 45
Immersify Plugin 45
Immersify Mesh Video 46
Video UV Controller 47
Immersify Cmd Config Mgr 47
Load Video by Config 48
Results 48
Specification of media playback systems 49
Advances over the State of the Art 49
Unreal Immersify Player **50**
Concept 50
Using the Unreal Immersify Player Plugin 50
Preparation 50
Scene Setup 51
Parameters and Options 52
Results 56
Specification of media playback systems 56
Advances over the State of the Art 56
PSNC 8K 3D Media Player **59**
Concept 59

Implementation 59
Results 64
Specification of media playback systems 65
Advances over the State of the Art 67
PSNC Cave Media Player **68**
Concept 68
Implementation 69
Results 73
Specification of media playback systems 74
Advances over the State of the Art 75
NVAB Dome Player **76**
Concept 76
Implementation 76
Results 78
Specification of media playback systems 78
Advances over the State of the Art 78
Task Status **80**
Expected Results 80
Obtained Results 80

1 Introduction

Immersify is founded on top of three main elements: the creation of immersive content, its exhibition in immersive environments, and the use of video codec tools for compression and display of the content in the target environments.

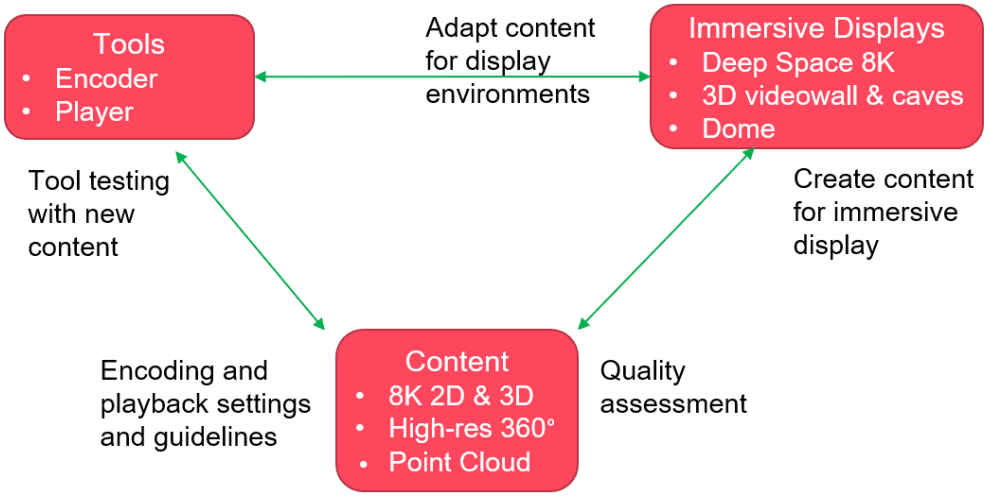


Figure 1: Immersify: tools, environments, and content for highly immersive applications

We use the video encoders developed in the project (WP4) to reduce the size of the original files coming from the content production (WP5) at the quality level required by the target use cases.

The compressed media files are then used as input by the media players developed in WP3 to display the content at the targeted immersive environments. The media player should be able to decode, render, and present ultra-high resolution video and spatial audio. The required components for these media players, including video decoders and video renderers, and ambisonic audio decoders and renderers, have been developed and optimized in WP3 in the context of Tasks: T3.1 *Decoder and Renderer performance optimization*, T3.2 *Decoder with VR specific features*, T3.3 *Multi-screen and scalable display*, and T3.4 *Immersive Audio*. Task 3.5 *Media player integration*, is dedicated to the integration of these media components into complete media players.

As the immersive content is being produced using a wide range of techniques and it targets a wide range of heterogeneous immersive environments, multiple media players have been developed from a common pool of media components available in a media SDK.

Task 3.5 addresses some of the main objectives of WP3 and of the overall project, that is to support multiple immersive display environments including 8K video walls, Deep Space 8K, cylindrical caves, and domes.

In this report we present the results of Task 3.5, including the implementation of new features for audio and video decoding and rendering, as well as their integration into the different media players developed in the context of the project.

2 Task 3.5 Description

2.1 Original Task Description

According to the Grant Agreement, the goals of "Task 3.5 - Media Player integration" are:

The main objective of this task is to integrate the enhanced HEVC decoder, video renderer, and 3D audio codec into a full-featured media player for immersive spaces. Different media players are envisioned in the project for different use cases: a) a PC media player for multi-screen (e.g. an array of curved screens forming a wide field of view curved display) including multi-instance playback synchronisation e.g. via external timecode, b) a media player for head-mounted-displays that supports both PC and mobile devices, c) a media player for multi-projection systems such as the AE Deep Spaces or Domes in planetariums supporting quad-buffered stereoscopic rendering and d) a VR media player integrated in the 3D engine Unity3D (targeting primarily Windows platform).

This task is organized into the following subtasks:

- *1) integration of the optimized decoder and renderer from T3.1, T3.2 and T3.3 into the target playback platforms, such as: media player based on "Media Player Classic BE" (SD), Deep Space 8K player (AE), the media server and 8K 3D network player (PSNC), Dome theater player of NVAB, and an Unity3D plugin wrapper.*
- *2) Porting the media player to multiple HMD displays including PC based (e.g. Oculus Rift, HTC Vive with Windows OS) and mobile-based ones (e.g. Samsung Gear VR with Android OS).*
- *3) Specification of reference systems that include information about the configuration of the hardware, system software, and interconnection with TVs and projectors.*

2.2 Deviations from the Description of Work

Compared to the original plan described above three changes were applied in order to better reflect the intended use cases of the project:

- **Focus on large immersive visualization not HMD or mobile**: Initially it was expected that the media players and associated video decoding and rendering tools can be ported, in addition to large screen systems based on powerful workstations and servers, to HMDs for desktop PCs and mobile devices. An underlying assumption was that there will be a new generation of HMDs with very high resolution and high video quality that can be used for consumption of highly immersive media. It turned out that the evolution of HMD technology was slower than anticipated and the available devices during the project time did not have the required technical specifications for highly immersive applications. Instead we decided to focus on large scale immersive environments such as the AEC Deep Space 8K, the 8K Caves and Video Walls at PSNC, or the 8K Dome at NVAB.
- **Extend the duration of the media player integration task**: Task 3.5 was originally scheduled to finish on month 24 of the project, and after that the resulting media players will be used for the demonstrations planned in WP5. As part of the project extension decided and agreed in 2019 (Amendment 2), Task 3.5 was extended until M30. This extension was required for performing additional development effort on features required by some of the target media players including interactive 360° with multi-GPU support, extended support for high-order ambisonics audio, and support for geometry corrections for curved screens.

- **Perform multiple integration iterations**: Instead of a single phase of media player integration in practice we did several rounds of development of new features and integrating them into different media players. At least three main phases occurred. The first phase included mainly the development of optimizations to the basic player components such as the HEVC decoder and the video rendering engine using efficient pixel formats and texture compression formats in order to reach the required performance for ultra-high resolution video playback up to 16Kp60. The second phase included new features for more responsive 360° video playback (decoupled video rendering) and multi-GPU rendering support. The final phase, described in detail in this report included mainly improvements for 360° video rendering (input and output projection formats) and high-order ambisonics audio playback.

2.3 Relevance of the Task to the Project

The work performed in Task 3.5 addressed the following two general project objectives:

- **To enable immersive media for multiple display environments and multiple devices**: VR and immersive media can be consumed in many environments and devices including multiple PC- and mobile-based HMDs, multi-display settings, and giant screen environments (dome, deep space, immersive cinema). Both single user and collective experiences (in Film and Art festivals, and cinemas) are being demanded by the industry. Immersify will provide media players and format conversion tools required to facilitate the consumption of immersive media in multiple environments and devices.
- **To support new ways in interactive experiences by providing the required tools for personalized and interactive non-linear storytelling**. The interaction capabilities of current VR media is very limited, in most cases confined to the exploration of a panoramic video with head rotation using HMDs. Production tools for more advanced user interaction [are] not available or are very limited. Immersify will provide tools for integrating high quality video in seamless media presentations that will allow content creators to develop non-linear storytelling where the user has the freedom to select different paths in the content.

A key component for making high quality immersive multimedia possible is the media player.

It should support very high resolution and high quality video (up to 16K and beyond). It should be able to play video in the different formats being used for immersive media, including high resolution 2D video, stereoscopic 3D video, 360° panoramic and interactive video, and point cloud renderings. In addition the media players for immersive media should support high quality audio formats, including large multi-channel (22.2) and high-order ambisonics with rendering to different channel layouts.

As immersive content is presented in a wide range of heterogeneous immersive environments, including large screen immersive planar environments, domes, caves, etc., it is desirable that the media player help in the adaptation of the content to the display environment. Adaptations include, among others, color conversions and geometry adaptations. Geometry and display adaptations are very relevant as they allow a single input file to be played on different display environments, such as high-resolution flat and curved screens, then reducing production costs and increasing interoperability by reducing fragmentation. Depending on the target viewing environment, adapted viewports with different viewpoints, FoVs, and resolutions can be extracted and then projected onto either regular flat or curved displays.

3 New Immersive Features

3.1 Introduction

Over the duration of the project, the features developed in Work Package 3 were included in a SDK created by Spin Digital called Spin SDK. The media players developed by Spin Digital are based on the Spin SDK, which integrates all these features to provide an immersive experience.

The set of features herein presented were developed after the Immersify deliverable "D3.2 - Report on Decoder and Video Rendering for VR - Part 2" in month 24, to increase the overall user interactivity with 360° videos. Hence, when considering the development of Task 3.5 - Media player integration, the Spin SDK was extended with more immersive features, such as:

- equirectangular projection
- curved screen support with cylindrical projection
- multi-device support with interactive 360° video
- 360° navigation with the Xbox controller
- new licensing mode
- HOA sound extension

All the components described herein are included in the Spin SDK release v. 3.0.

3.2 Equirectangular Projection (ERP)

Since equirectangular projection (ERP) is one of the most common 360° video formats, its support expands the immersive players' capabilities based on the Spin SDK. Moreover, the cube map format generates border artifacts due to the GPU interpolation filters over the edge of non-adjacent faces. This problem does not happen with the equirectangular projection. The support for equirectangular is performed in two steps in the Spin SDK. First, each frame is tagged as plane (default), equirectangular, or cube map by the Spin Digital HEVC decoder (SpinDec). Later, the Spin Digital video renderer (SpinRender) is responsible for constructing the viewport accordingly, by mapping each pixel of the viewport to the corresponding sample in the equirectangular 2D scene. If the corresponding sample location does not match a pixel position, bilinear filtering interpolation is used to generate the position sample value.

3.3 Curved Screen Support with Cylindrical Projection

When considering 360° video playback in flat 2D screens, the immersive experience may be limited due to distortions when mapping a section of the 360° frame in a 2D plane. By supporting curved screens, the overall experience is improved since the distortion effects are reduced, providing a more natural result. The Spin SDK supports curved screens by providing the cylinder projection in addition to the plane projection. In Figure 2, a top view of a 360° picture on a sphere is shown, where it is mapped into a flat and curved screen using the plane and the cylinder projection, respectively.

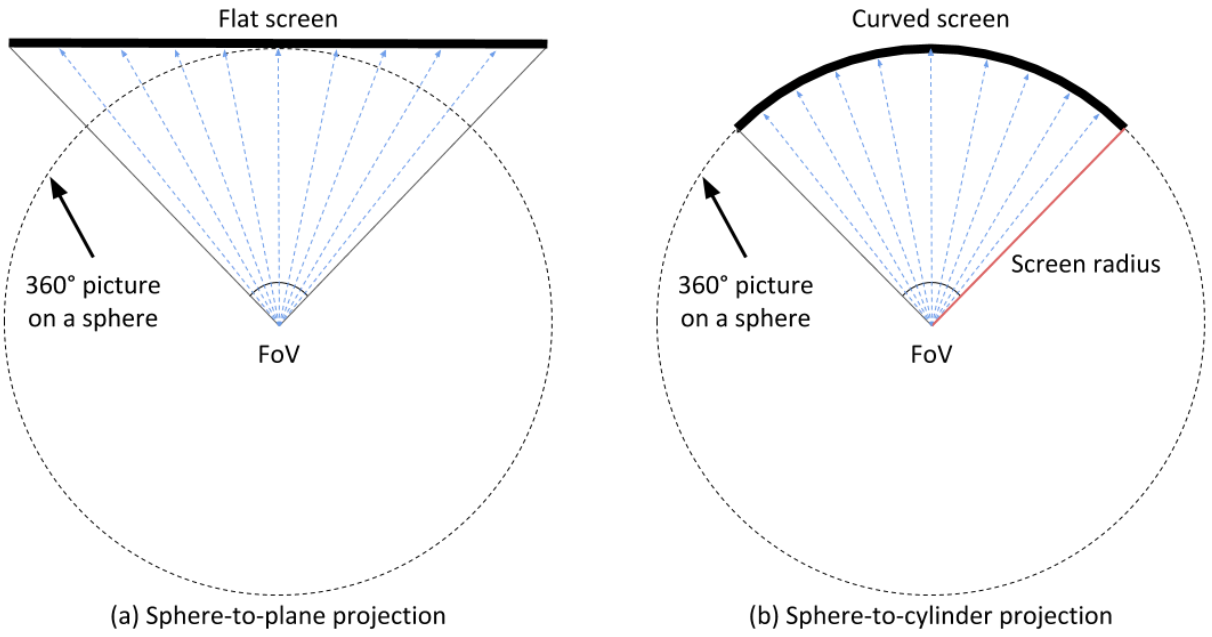


Figure 2: Sphere-to-plane (a) and sphere-to-cylinder (b) projections targeting flat and curved screens, respectively

The plane projection consists of mapping a section (or viewport) of the surface of a sphere to a flat surface, as shown in Figure 2a. This portion is determined by the Field of View (FoV). The vertical FoV is set to 70° by default. The horizontal FoV is accordingly derived based on the display aspect ratio. Nevertheless, the FoV can be modified dynamically by the user through the zoom in and zoom out operations. As shown in Figure 2a, the edges of the flat screen have less pixel density (less blue arrows) from the sphere, thus leading to a distorted or stretched image depending on the sphere radius. When considering cylinder projection, see Figure 2b, the edge of curved screens has a higher pixel density than in comparison with flat screens. Therefore, the distortions present in the edge of the flat screen are going to be significantly reduced if the screen radius is the same as the sphere radius in curved screens. Hence, the cylinder projection on curved screens contributes to a more natural visual experience for 360° videos. On Spin SDK, the output projection is selected by the *eDispGeom* parameter of the SpinRender, where the options are: *SE_GEOM_None* (no mapping), *SE_GEOM_Plane*, and *SE_GEOM_Cylinder*.

3.4 Multi-device Support with Interactive 360° Video

Although the feature reported in the deliverable D3.2 already supported connecting multiple displays to form a single immersive giant screen, there was no interactive capability for omnidirectional videos. Hence, the user could not perform 360° scene navigation on multiple screens. The feature reported herein expands the SpinRender in order to provide such behavior.

The previous implementation of multi-device has been done for regular 2D video in a fixed rectangular layout. This model provides the capability of the media player to combine multiple screens (projectors, TVs, or computer monitors) into a single large display. Hence, a single large resolution 2D video can be displayed using a tiled rendering mode with multiple GPUs and multiple screens. Note that although a single virtual display can be created (e.g., with NVIDIA Mosaic), the DirectX 12 API does not support rendering across GPUs, and DirectX 11 API does not provide the required high performance. Therefore, in the tile rendering mode, the renderer divides the input

frame into several tiles before uploading data to the texture. Observe that the maximum theoretical input resolution is sixteen 16384×16834-pixel blocks in any arrangement. Sixteen, because, currently, on Windows 10, only up to 16 displays are supported, and the Microsoft DirectX API has a limit of the 2D texture width and height of 16384 pixels per GPU. However, the real maximum input resolution is a function of the frame rate and other hardware specifications (e.g., PCIe bandwidth, disk I/O speed, among others). The disadvantage of this method is that since each GPU has only part of the frame, interactivity (e.g., pan and zoom) is not possible.

In order to provide interactivity with 360° scene navigation on multiple screens, the new Spin SDK introduces an additional tiled rendering mode in which the whole frame is copied to each device (GPU). Each GPU receives and renders the entire video frame but presents only the portion of the video within the viewport. The region of the video displayed on each device has to be re-computed every time that there is a viewport update. The only limitation is due to the Microsoft DirectX API 2D texture. Accordingly, in the new tiled rendering mode, the maximum input video frame resolution is 16384×16834 (16K×16K) pixels. Hence, the 2D video frame, which represents a 360° video frame, can't have a resolution higher than 16K pixels in the horizontal and vertical direction.

Nevertheless, the maximum output dimension can be higher than 16K according to the number of devices present in the system. For example, the NVIDIA Quadro M4000 GPU can support each DisplayPort 1.2 connector with a maximum resolution of 4K at 60 Hz. Therefore, for a system with four of those GPUs, a maximum of sixteen 4K pixel blocks in any arrangement is possible. When "in any arrangement" is mentioned, it means that the sixteen displays can be any matrix of displays like 1×16, 4×4, or 2×8. Moreover, flexible displays arrangements can be used, for example, in a cylinder cave, where the number of displays per column is higher than per row.

It is important to notice that in most of the use-cases, the viewport resolution is not the same as the input resolution since just a fraction of the 360° video is being displayed. In Table 1, a summary of the characteristics of each rendering mode is presented, where the single-device rendering mode considers only one device (GPU). In the single-device rendering mode, as well as in the interactive multi-device rendering, the input frame size cannot be more than 16384 pixels in either horizontal or vertical directions.

In what concerns the Spin SDK implementation, the rendering mode is set by the SpinRender parameter *eDeviceMode*, where the values *SE_DEVICEMODE_Single*, *SE_DEVICEMODE_Tiled*, and *SE_DEVICEMODE_Tiled2* correspond to single-device rendering mode, multi-device rendering mode, and interactive multi-device rendering mode, respectively.

The user interaction for 360° scene navigation is provided by two methods: messages or a callback function.

Table 1: Comparison of the supported features by single device, multi-device rendering and interactive multi-device

	Single device rendering	Multi-device rendering	Interactive multi-device rendering
360° scene navigation	✓	X	✓
Max. number of GPUs	1	4	**4**
Max. number of displays	4[1]	16	**16**
Max. input resolution[2]	16384×16834[2]	16×16384×16834[2]	**16384×16834[2]**
Max. output resolution	4×4K[3]	16×4K[4]	**16×4K[4]**

3.4.1 Messages

One way to control the render engine is performed by sending messages through the SpinRender API. Hence, the application sends messages, such as change of color format, zoom in and out, viewport changes, among others, and the rendering thread is responsible for carrying out the desired action. This behavior is achieved by storing a queue of messages which are checked every time a frame will be rendered. In decoupled rendering mode, the maximum latency will be the display frequency plus the latency of the application in detecting the movement or zoom and sending the message.

3.4.2 Callback function

In the Spin SDK, a callback function present in the SpinRender parameter can be set in order to provide the position and FoV of the viewport. The returned structure of the callback function should contain the viewport FoV, yaw (azimut) and pitch (elevation) rotation. Inside the library, the viewport pose and FoV will be queried through the callback function every time the render engine renders a frame. Accordingly, the maximum latency in the decoupled rendering mode from this method is the display frequency plus the latency from updating the viewport pose, which is queried by the callback function.

3.5 360° Navigation with the Xbox Controller

With the advent of the new Spin SDK with the possibility to control the viewport pose and FoV from a callback function, it was possible to use a better way to navigate in a 360° scene. Previously, in the Spin Digital media player, the only way to change the viewport was with the mouse. The mouse movements were captured in an auxiliary window in which messages were sent to the SpinRender to

[1] 4 display outputs have been tested but in principle 6 display outputs are possible when using GPUs such as the AMD WX9100 which supports 6x Mini-DisplayPort 1.4

[2] Maximum theoretical limit according to the DirectX API 2D texture limitations, the real maximum input resolution can be lower since it is a function of the frame rate and hardware specifications.

[3] With DisplayPort 1.2. Also, Windows 10 can not support desktop resolution higher than 16384×16834 on NVIDIA Mosaic or AMD Eyefinity.

[4] With DisplayPort 1.2. Moreover, Windows 10 can only support up to 16 displays in the same system.

update the viewport. This approach is not practical since several mouse movements (clicks and drags) are needed to rotate the viewport, which sometimes can be burdensome. On the other hand, the Xbox controller, which is used mostly by gamers, can be a more intuitive, practical, and natural way to navigate into 360° videos.

Therefore, the Xbox controller support was added to the Spin Digital media player and updated the viewport via the callback function. In this case, a separated CPU thread is responsible for querying and keeping tracking of the Xbox controller user interactions. Inside the Spin SDK library, the render engine calls the given callback function and gets the latest viewport FoV, yaw, and pitch. In Figure 3 in combination with Table 2, the Xbox controller supported functions are described.

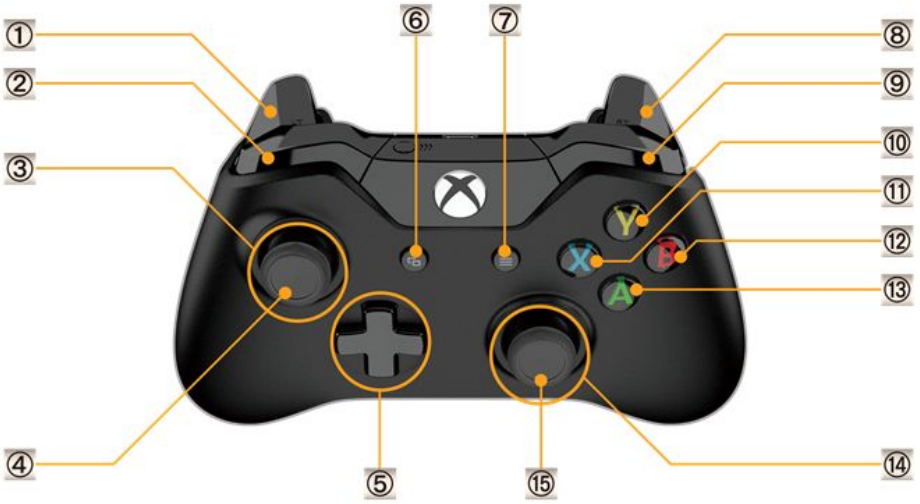


Figure 3: Xbox controller buttons and thumbstick assignments

Table 2: Xbox controller functions

Button or Thumbstick	Function
1	Increase the sensitivity of the thumbstick
3	Zoom in and out
4	Reset zoom factor
8	Decrease the sensitivity of the thumbstick
14	Yaw (azimut) and pitch (elevation) rotation
15	Reset rotation

3.6 New Licensing Mode: needed for the release of 3D engines plugins

Description of software protection scheme

In order to protect the IP in the form of binaries distributed to final users, and to enforce license restrictions, the Spin SDK uses the protection and licensing methods provided by the CodeMeter technology from Wibu Systems[5]. The binary protection against reverse engineering and debugging is applied with the help of the AxProtector software from CodeMeter which encrypts the Spin SDK binaries. For decrypting and running the binaries, a valid license is needed. In addition to this high-level software protection, a more detailed internal license checking mechanism is also implemented. It checks the available license and its features against the features that the Spin SDK library is currently trying to use. If a requested feature is not available, the application exits with an error message. Examples for such license features are the allowed maximum resolution, pixel formats and type of output devices.

The license itself is stored on a hardware key in the form of a USB stick, called CmDongle. The license is bound to the dongle. The dongle (and with it the license) can be moved freely from one machine to another. This is the default distribution method that is currently used for Spin SDK (and Spin Player). Alternatively, the license can also be stored inside a software container, called CmActLicense. This approach can be used if the hardware key is not practical for some reason. However, the CmActLicense has to be activated by hand before use and is then bound to the specific system.

Limitations of USB hardware key software protection and new licensing method

Although the CodeMeter licensing system works well, it has limitations for certain use cases. One of these problematic use cases is the 3D engine integrations (Unity3D and Unreal) that are described in Section 6 and 7. Here, the use of a hardware stick or the use of a machine-bound license which has to be activated is not practical as the 3D engine plugins will be released as free-of-charge software downloads from the asset stores of the 3D engines.

As an alternative, a new licensing method was integrated into Spin SDK. It uses the Sentinel RMS licensing solution from Thales[6]. It provides more flexible licensing methods. One of the methods is the so-called license *leasing*. It automatically fetches a license from a license server and stores it locally where it is valid for a certain period of time, e.g. one month. Before it becomes invalid, the license is renewed automatically. Internet connection is required only for the license updates.

Alternatively, the license can also be used directly in the cloud without the need to store it locally. This is called Cloud Connected licensing. To use the license, the user only needs to provide some credentials, but a constant Internet connection is required to run the software. With these additional licensing modes, easy and scalable software distribution as it is required for the 3D engine plugins is possible.

In the new Spin SDK, these different licensing methods are now available to be used. There are three different sets of binaries provided. Each one is compatible with a specific license method. They can be distinguished with their filename suffix:

- "_cm" binaries are used for the CodeMeter licenses, e.g. CmDongle.
- "_rms" binaries are used for the Sentinel RMS licenses, e.g. RMS Leased License.

[5] CodeMeter. https://www.wibu.com/products/codemeter.html
[6] Sentinel RMS. https://docs.sentinel.gemalto.com/softwareandservices/rms/

- "_scl" binaries are used for the Sentinel Cloud Connected licenses.

Spin SDK now also provides a utility tool called "spinlicensetool". It takes configuration options for the preferred license server and user credentials (if required) and creates a "licenseconfig.txt" file. This file is used by the Spin SDK applications and controls from where the license is taken. The use of this tool is only required in certain use cases. If the CodeMeter binaries are used and no specified license selection is required, no interaction is needed. This corresponds to the usage in previous versions of Spin SDK. For other licensing modes like the Sentinel leased or cloud connected modes, some configuration is required. However, in the case of the basic version of the 3D engine plugins, special configuration would not be required by the end-user because the software package can already contain a preconfigured license setting.

Application of the new licensing method to the 3D engine plugin distribution

The Immersify 3D engine plugins will be distributed using the asset stores of the corresponding engines such as Unity Asset Store[7] and Unreal Marketplace[8] which are the most common way of distributing plugins for the 3D engine developer communities. The plugins will be provided for free and they will include an evaluation license of the Spin Digital HEVC decoder for large resolution video.

The plugin will be licensed using the RMS Leased License mode, or Sentinel Cloud Connected mode. These license modes will allow the users to use the application after download with minimal configuration, and compared to the full commercial license will have some restrictions (reduced maximum resolution and color format, on-screen mark, and time limitation). A full commercial license can be acquired from Spin Digital, and the license methods allow a simple license update procedure. More details of the commercial exploitation strategy are described in the deliverable D2.4 "Final Plan for Use and Dissemination of Foreground".

3.7 HOA Sound Extension

The HOA decoding library developed in Task 3.4 on "Immersive Audio" and reported in D3.2 on "Report on Video Decoding and Rendering for VR - Part 2" specified some standard loudspeaker layouts selectable by the user, more specifically: binaural, stereo, quad, 5.1, 7.1, and 22.2. Therefore, the target use cases were very limited, since other loudspeaker configurations, including the custom ones, were not supported.

In addition, the library was developed to provide support for up to 3rd-order ambisonics. Although this order is high enough for the majority of multichannel sound systems, we also noticed that higher order decoders provided a finer sound localization in the 3D space. These beyond-3rd-order decoders are in fact becoming more popular among state-of-the-art audio plugins, which already support up to 7th-order HOA[9],[10],[11].

[7] Unity Asset Store https://assetstore.unity.com/
[8] Unreal Marketplace. https://www.unrealengine.com/marketplace/en-US/store
[9] D. Uldrich, IEM plug-in Suite, AllRADecoder: https://plugins.iem.at/
[10] A. Politis, L. McCormack, Spatial Audio Real-time Applications (SPARTA), AmbiDec: http://research.spa.aalto.fi/projects/sparta_vsts/plugins.html
[11] M. Kronlachner, Ambisonic plug-in Suit, (ambiX): http://www.matthiaskronlachner.com/?p=2015

3.7.1 Custom Loudspeaker Layouts

For custom loudspeaker layouts, the constellation (distance, gain, spatial position of each loudspeaker) and the decoder matrix can be specified in a *JavaScript Object Notation* (JSON) file. This format has been adopted by most of the HOA plugins as the method for importing and exporting decoder matrices and loudspeaker layouts.

The open-source C++ parser library named *jsonccp*[12] was integrated in our HOA decoding library to import the layout information and decoder matrices generated by external tools. Although several tools are available online, we recommend the open-source IEM *AllRADecoder* plugin because of its intuitive and easy-to-use user interface.

The main attributes of a JSON file are described next:

- **"ExpectedInputNormalization"** accepts two options: n3d or sn3d. Since the Spin Digital HOA library only supports the sn3d normalization, in the case of n3d the library converts the matrix coefficients from n3d to sn3d.
- **"Weights"** defines the weighting method of the decoder, which can be: none, inPhase/inphase, maxrE/maxre, or Dual/dual. Because of its good performance in terms of sound localization, it is recommended to select the dual-band method.
- **"WeightsAlreadyApplied"** can be false or true:
    - If false, the matrix coefficients are the ones corresponding to the basic decoder ("Weights": none) and the Spin Digital HOA decoder will convert internally the coefficients from basic to the selected weighting method.
    - If true, the coefficients are the ones corresponding to the selected weighting method and the Spin Digital HOA decoder will not perform any conversion.
- **"SubwooferChannel"** defines the channel number of the subwoofer. For example, if the layout is standard 5.1, this number will be 4, that is: "SubwooferChannel": [4]. If the layout includes more than one subwoofer channel, this attribute will be a vector, for example: "SubwooferChannel": [4, 7].
- **"Matrix"** specifies the decoder coefficients. Since the subwoofer feeds are not created from ambisonics decoding, the matrix only contains *N* rows, where *N* denotes the number of loudspeakers in the layout excluding the subwoofers.
- **"Routing"** Defines the output routing as a vector of integers. Each loudspeaker feed will be routed to the specified channel number (see below).
- **"Loudspeakers"** describes the loudspeaker constellation. For each loudspeaker the following attributes are provided:
    - "Azimuth" <degrees>: azimuth coordinate.
    - "Elevation" <degrees>: elevation coordinate.
    - "Radius" <meter>: distance between the loudspeaker and the listener.
    - "Channel" <integer>: channel number.
    - "Gain" <float>: gain expressed as dimensionless unit.

[12] JsonCpp, A C++ library for interacting with JSON: https://github.com/open-source-parsers/jsoncpp/tree/1.9.0

A JSON file example for a 5.1-channel system is given below:

```
{
  "Decoder": {
       "Description": "A 2nd-order ambisonics decoder for 5.1 sound.",
       "SubwooferChannel": [4],
       "ExpectedInputNormalization": "sn3d",
       "Weights": "Dual",
       "WeightsAlreadyApplied": false,
       "Matrix": [
       [0.199, 0.145, 0.073, 0.140, 0.144, 0.000, 0.009, 0.016, -0.015],
       [0.199,-0.145, 0.073, 0.140,-0.144, 0.000, 0.009, 0.016, -0.015],
       [0.130, 0.000, 0.073, 0.112, 0.000, 0.000  0.028, 0.016,  0.097],
       [0.379, 0.212, 0.073,-0.213,-0.041, 0.000,-0.019, 0.016, -0.025],
       [0.379,-0.212, 0.073,-0.213, 0.041, 0.000,-0.019, 0.016, -0.025],
       ],
       "Routing": [1, 2, 3, 4, 5, 6]
  },
  "LoudspeakerLayout": {
       "Loudspeakers": [
       {
       "Azimuth": 30.0, "Elevation": 0.0, "Radius": 1.0,
       "Channel": 1, "Gain": 1.0
       },
       {
       "Azimuth": -30.0, "Elevation": 0.0, "Radius": 1.0,
       "Channel": 2, "Gain": 1.0
       },
       {
       "Azimuth": 0.0, "Elevation": 0.0, "Radius": 0.8659999966621399,
       "Channel": 3, "Gain": 0.8659999966621399
       },
       {
       "Azimuth": 15.0, "Elevation": 0.0, "Radius": 1.0,
       "Channel": 4, "Gain": 1.0
       },
       {
       "Azimuth": 110.0, "Elevation": 0.0, "Radius": 1.0,
       "Channel": 5, "Gain": 1.0
       },
       {
       "Azimuth": -110.0, "Elevation": 0.0, "Radius": 1.0,
       "Channel": 6, "Gain": 1.0
       }
       ]
  }
}
```

3.7.2 Rotation and Zoom of up to 7th Order

Besides the supported loudspeaker layouts, another main limitation of the HOA library developed in Task 3.4 was the lack of support for ambisonics sound beyond 3rd order. Although the library could accept audio files higher than 3rd order, it could only process (decode, rotate, and zoom) the first 16 channels.

The HOA library was enhanced to also allow for up to 7th-order decoding, including rotation and zoom. The recursive rotation method of Ivanic and Ruedenberg[13],[14] was implemented using as a reference the Daniel Rudrich's implementation included in the *AllRADecoder* plugin. Ivanic and Ruedenbergs's algorithm has also been adopted in other plugins, such as Sparta (*AmbiDec*)[15] and *ambiX*[16].

The enhancement of the zooming algorithm was quite simple, since it only required the inclusion of 4th-, 5th-, 6th-, and 7th-order mono decoder matrices pointing at the front where the screen is placed.

It is also worth mentioning that, from the performance point of view, these improvements are still light enough and do not compromise the real-time operation of the audio render engine.

[13] Ivanic, J., Ruedenberg, K. (1996). Rotation Matrices for Real Spherical Harmonics. Direct Determination by Recursion. The Journal of Physical Chemistry, 100(15), 6342-6347.
[14] Ivanic, J., Ruedenberg, K. (1998). Rotation Matrices for Real Spherical Harmonics. Direct Determination by Recursion Page: Additions and Corrections. Journal of Physical Chemistry A, 102(45), 9099-9100.
[15] Aalto University (2020). Spatial Audio Real-time Applications (SPARTA): http://research.spa.aalto.fi/projects/sparta_vsts/plugins.html
[16] M. Kronlachner (2014). Ambisonic Plug-in Suite: http://www.matthiaskronlachner.com/?p=2015

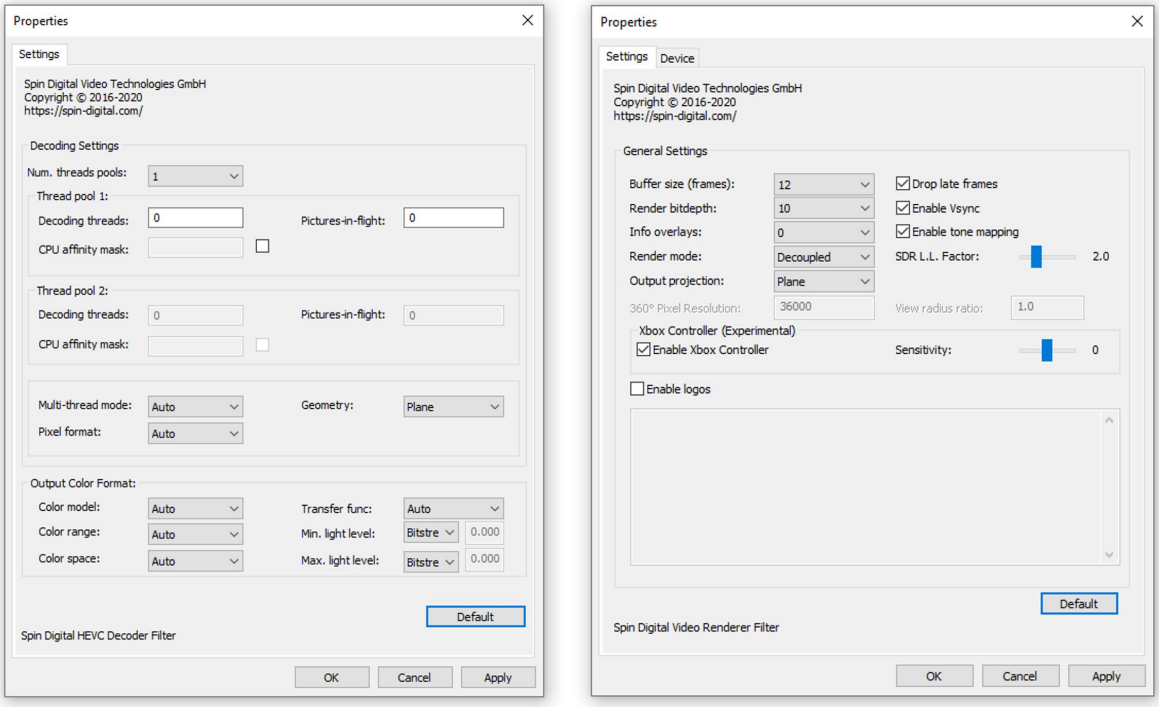
4 Spin Digital SDK and Media Players

4.1 Concept

Over the Work Package 3, several features were developed, structured, and integrated by Spin Digital into an SDK called Spin SDK. In what concerns the media player integration of the Immersify project, the main Spin SDK modules included are the Spin Digital HEVC Decoder (SpinDec) and Spin Digital Video Renderer (SpinRender). Hence, the organization of the developed features in an SDK allowed the Immersify partners to build their media players, e.g., Immersify Video Player, which is presented in Section 5.

The Spin Digital media player, Spin Player, is based on the DirectShow framework. Hence, for each module of the Spin SDK, a DirectShow filter is created in order to interact and configure the Spin SDK modules. All the Spin Digital DirectShow filters are listed as Spin Player's internal filters. They can be accessed via View → Options → Internal Filters. For each filter, a different set of configuration options are available.

Another option to access the Spin Digital DirectShow filters during playback is with a right-click on the main Spin Player window to open the context menu → Filters. The currently loaded filters are then presented, and the property pages of the shown filters, e.g., SpinDec filter and SpinRender filter, can be accessed by clicking them in this Filters menu during playback. In Figure 4, the settings of the Spin SDK modules can be configured via the DirectShow property pages of the SpinDec and SpinRender filters.



(a) SpinDec settings property page (b) SpinRender settings property page

Figure 4: Spin Digital DirectShow SpinDec (a) and SpinRender (b) property pages

The features developed in the Spin SDK and, consequently, the player integration has happened in three phases, where each phase targets a specific use-case:

1. Integration Phase 1: 16K video playback;
2. Integration Phase 2: Multi-device and responsive 360° playback;
3. Integration Phase 3: Immersive and interactive 360° video with spatial audio.

As follows, a short survey will be presented for each phase.

4.2 Implementation

4.2.1 Integration Phase 1: "16K video playback"

The progress reported of this phase in the Deliverable 3.1 - Report on Decoder and Video Rendering for VR - Part 1 aligns mainly to objective 1 of the Immersify project: "To improve the quality of immersive media using advanced compression technology". The work performed addresses this objective by improving video playback performance, to allow much higher resolution (up to 16K), frame rates, and pixel quality, on current and next-gen playback platforms.

The features developed during this phase can be summarized as:

- **Memory-efficient pixel format:** bit-aligned support offers the decoder and rendering capability to access the video frame stored in the memory more efficiently, which reduces the needed bandwidth and required memory size.
- **Compressed texture format:** the compressed texture format BC4 support allowed the Spin SDK to reduce the required bandwidth to transmit a 10-bit depth 16K video at 60 fps below the PCIe bandwidth limits.
- **AVX512:** the support for Intel AVX512 SIMD instructions allowed to improve the HEVC decoder performance in the Spin SDK.
- **Thread pool extensions for NUMA:** the support to schedule the frame groups into a single NUMA node with thread pools provides better performance for multi-socket machines.

These features were integrated into the Spin SDK 2.0, which was released in August 2018. The Spin Player v2.0 was released in March 2019, with all the Spin SDK 2.0 features integrated. In the Spin Player decoder settings, see Figure 4a, it is possible to configure the thread pools as well as to choose the pixel format (memory-efficient or compressed texture). The AVX512 is used automatically accordingly with the existing hardware; thus, no additional settings are required. In Table 3, a summary of the integration phase 1 is presented.

Table 3: Summary of integration phase 1: 16K video playback

Item	Name	Release Date
SDK	Spin SDK 2.0	August 2018
Media Player	Spin Player v2.0	March 2019
Deliverable	D3.1 - Report on Decoder and Video Rendering for VR - Part 1	September 2018
Tasks	Task 3.1: Decoder and Renderer Performance Optimization Task 3.2: Decoder with VR Specific Features	
Features	• Memory-efficient pixel format • Compressed texture format • AVX512 • Thread pool extensions for NUMA	

4.2.2 Integration Phase 2: "Multi-device and responsive 360° playback"

The features developed under this phase were focused on allowing two main use-cases: responsive 360° playback and multiple rendering devices support. The work was reported in the Deliverable 3.2 - Report on Decoder and Video Rendering for VR - Part 2 from September 2019. In what concerns responsive 360° playback, a new rendering mode was created where the rendering engine was "decoupled" from the frame upload engine. This new rendering mode allowed that viewport pose changes were updated at the display frequency instead of the video frame rate. The other main use-case provides multiple rendering devices support, where multiple displays output of multiple GPUs were combined to form a large screen display. With the advent of this new rendering mode, video playback with 16K display support at 60 frames per second becomes a reality.

Hence, the features developed during this phase can be summarized as:

- **Decoupled 360° video rendering:** the low-latency viewport poses updates enhanced the interactive playback experience for omnidirectional videos, where the rendering frame rate should be decoupled from the video frame rate.
- **Multi-screen display:** multi-screen refers to the capability of the media player to combine multiple screens (projectors, TVs, or computer monitors) into a single large display.

Both features were integrated into SpinRender in the Spin SDK 2.2, which was released in November 2019. The Spin Player v.2.1, which was also released in November 2019, was upgraded with the new SpinRender. The decoupled rendering mode can be selected by the "Render mode" option of the SpinRender DirectShow property page (see Figure 4b). On the other hand, the multi-screen display settings are under the "Device" tab of the property page. In Figure 5, the device settings of the SpinRender property page is shown. When the "Device mode" is "Tiled", a multi-screen display can be set up accordingly with the "Outputs", "Topology" and "Render Devices" settings, which are shown in Figure 5.

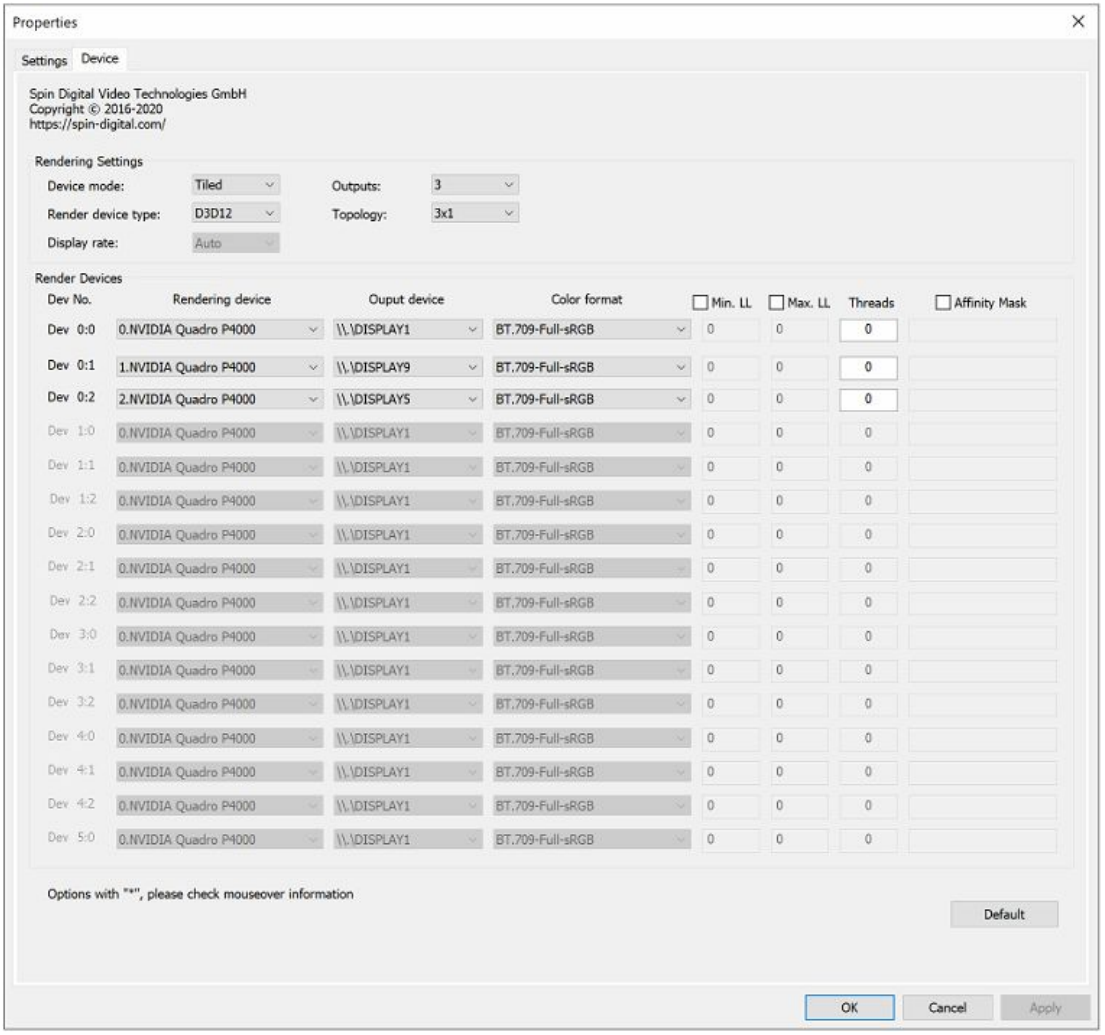


Figure 5: Device settings of SpinRender DirectShow property page

In Table 4, a summary of the integration phase 2 is presented. It is important to note that although part of Task 3.4 - Immersive Audio was reported in the Deliverable 3.2, there was no player integration. Hence, this task integration will be described in the next section.

Table 4: Summary of integration phase 2: Multi-device and responsive 360° playback

Item	Name	Release Date
SDK	Spin SDK 2.2	November 2019
Media Player	Spin Player v2.0	November 2019
Deliverable	D3.2 - Report on Decoder and Video Rendering for VR - Part 2	September 2019
Tasks	Task 3.2: Decoder with VR Specific Features Task 3.3: Multi-screen and scalable display	
Features	• Decoupled 360° video rendering • Multi-screen display	

4.2.3 Integration Phase 3: “Immersive and interactive 360° video with spatial audio”

The final integration phase aims to provide an immersive and interactive player for omnidirectional videos. The features which were target to this phase were already detailed in Section 3 - New Immersive Features. Moreover, the work reported in the herein Deliverable 3.3 - Report on Immersive Media Player Integration concludes the Task 3.2 - Decoder with VR specific features, Task 3.3 - Multi-screen and scalable display and Task 3.4 - Immersive audio. The summary of the latest features can be found below:

- **Equirectangular projection:** the support for this format expands the player’s capabilities of possible 360° input videos. Notice that equirectangular is a widespread format.
- **Curved screen support with cylindrical projection:** with the cylinder output projection, it was possible to support curved screens for a more immersive scenario.
- **Multi-device support with interactive 360° video:** the additional tiled mode enabled the interactivity into multi-screen systems.
- **360° navigation with the Xbox controller:** the support for the Xbox controller allowed the user a more convenient way to interact with omnidirectional videos.
- **New licensing mode:** additional licensing modes which do not require a physical dongle were required for the integration with 3D engine plugins to be released as free-of-charge software downloads from the asset stores of the 3D engines.
- **HOA sound extension:** a new audio rendering module was enhanced with support for 1) custom layouts and 2) up to 7th -order ambisonic decoding, including rotation and zoom.

All those features were included in the Spin SDK 3.0. The Spin Player v.2.2 encloses the new Spin SDK 3.0. However, the “New licensing mode” and “HOA sound extension” were not included since they are still experimental. Nevertheless, all the Spin SDK 3.0 features are involved in a new Spin Digital player called “streamplay”. This player is not based in the DirectShow framework and provides a low latency streaming module, which is mainly used as a live media player. Furthermore, since streamplay is conceived in a custom mini framework it is more practical to test and validate new features on it. *Streamplay* is a command-line player, where the internal Spin SDK modules are configured by command-line options. Spin Digital is building a new media player for live and immersive media on top of the low-level *streamplay* player, more details of the exploitation strategy are provided in the deliverable “D2.4 - D2.4 – Final Plan for Use and Dissemination of Foreground”

In what concerns the Spin Player v.2.2, the equirectangular format is set via the SpinDec property via the “Geometry” option (see Figure 4a), where the possible options are: Plane, Equirectangular and Cubemap. The Plane geometry format is the default option and is set for regular 2D videos. It is important to note that this information does not affect the video decoding process itself. It merely informs the video geometry of the frames. In the streamplay, the same options are controlled by the following command-line option:

- **--geom <plane|erp|cmp>** Geometry of input video

Here, *erp* and *cmp* mean Equirectangular (ERP) and Cubemap (CMP) formats, respectively. At runtime, during playback, the geometry format can be changed with the keyboard shortcut P.

The cylindrical projection and Xbox controller are configurable in the Spin Player via the SpinRender property page, as shown in Figure 4b via the “Output projection”, “360° Pixel Resolution”, “View radius ratio” options and “Xbox Controller” settings, respectively. On streamplay, the command-line

options related to the output projection are part of the render options (-r <render_opts>, --render <render_opts>) and are listed below:

- **dispgeom=<cyl|plane>** Indicates the display geometry:cylindrical (cyl) or planar (plane). Default=plane.
- **circres=<int>** Screen curvature in pixels. It represents the curvature as the pixels that the screen would have for a full cylinder of monitors. If the screen width is 11520 pixels and provides a field of view of 120° for viewing distance equal to the screen radius, then circres is three times 11520 pixels. Default = 36000.
- **radrat=<float>** View radius ratio (screen radius / viewing distance). If radrat=1, then the viewing distance is equal to the radius of curvature, and the 360° picture on the sphere is aligned to the display curvature. Default = 1.0.

Moreover, on the streamplay, the Xbox controller is detected automatically.

In the SpinPlayer, the additional "Device mode" option, which enables multi-device interactive 360° video support, is called Tiled360. This option can be selected in the "Device" tab, as it is shown in Figure 5. On the other hand, in the streamplay, this mode is called tiled2, and it is part of the render options:

- **devmode=<single|clone|tiled|tiled2|multi|alt>** Rendering device mode

The HOA-related options are only present in the streamplay, they are described below:

- **--isambix <0|1>** Flag to indicate if the file is Ambix (ACN/SN3D) or not. If 0, then the file is rendered as it is. For ambisonics decode this flag has to be set to 1. Default: 0
- **--mutelfe <0|1>** Flag to mute the input LFE signal(s). Default: 0
- **--gainlfe <real>** Initial gain in dB for the input LFE signal. Default: 0 dB
- **--json <file>** JSON file name. Only applied when layout is set to custom.
- **--zoomdepth <0.0-1.0>** Zoom depth (or the contribution of the foreground source to the sound field) when zooming in. If 1, then the zoom effect is very strong. If < 1, the zoom effect is weaker. If 0, there is no zoom effect. Default: 1.0
- **--rotalg <1|2>** Rotation algorithm:
  - 1 : Initial algorithm implemented in the baseline framework, is valid for up to 3rd order. If the input file contains higher orders, sound rotation and zoom will be disabled. This option will be deprecated
  - 2 (default) : [Experimental] Algorithm of Ivanic and Ruedenberg, valid for up to 7th order
- **--decmethod <basic|maxre|inphase|dual>** Decoding method for the supported loudspeaker layouts:
  - basic : Good for frequencies < 380 Hz
  - maxre : Good for frequencies > 380 Hz
  - inphase : Wider sweet spot but poorer source localization. Good for large audiences
  - dual (default) : Basic decoder for low frequencies and max-rE decoder for mid-high frequencies. More precise source localization but narrower sweet spot

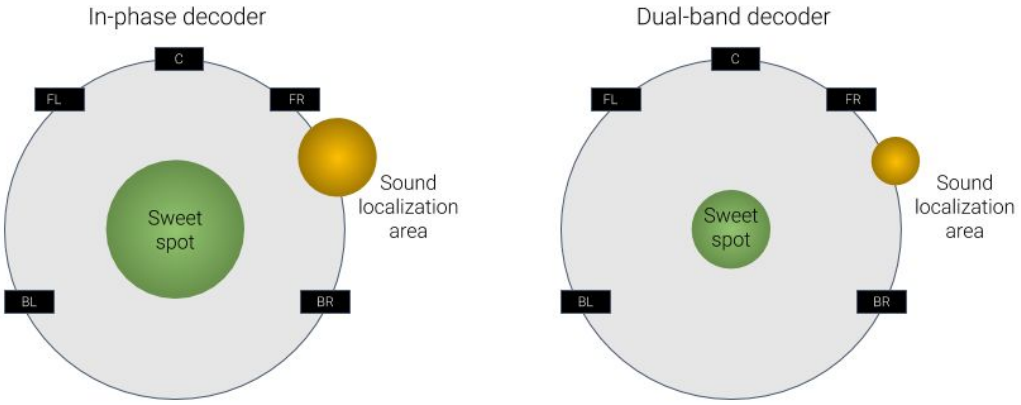


Figure 6: In-phase decoder (left) versus dual-band decoder (right) in terms of sweet spot and sound localization for a 5.1-channel layout

- **--layout <mono|binaural|stereo|quad|5.1|5.1_90|7.1|22.2|custom>** Loudspeaker layout:
  - mono
  - binaural
  - stereo (default) : 30, -30
  - quad : 45, -45, 135, -135
  - 5.1 : 30, -30, 0, LFE, 110, -110
  - 5.1_90 : 30, -30, 0, LFE, 90, -90
  - 7.1 : 30, -30, 0, LFE, 90, -90, 135, -135
  - 22.2 : NHK Layout
  - custom : Custom layout from JSON file
- **--dist <float > 0>** Distance in meters between the listener and the front-left (FL) or front-right (FR) loudspeakers for near-field compensation (NFC). The other distances are computed automatically based on the standards. This parameter shall be set for the standard loudspeaker layouts (except binaural) especified in --layout. For custom layouts, the distances are taken from the JSON file. Default: 1 meter

Finally, a summary of phase 3 integration is presented in Table 5.

Table 5: Summary of integration phase 3: Immersive and interactive 360° video with spatial audio

Item	Name	Release Date
SDK	Spin SDK 3.0	June 2020
Media Player	Spin Player v2.2 streamplay	June 2020
Deliverable	D3.3 - Report on Media Player Integration	June 2020
Tasks	Task 3.2: Decoder with VR Specific Features Task 3.3: Multi-screen and scalable display Task 3.4: Immersive audio Task 3.5: Media player integration	
Features	• Equirectangular format • Curved screen support with cylindrical projection • Multi-device support with interactive 360° video • 360° navigation with the Xbox controller • New licensing mode • HOA sound extension	

4.3 Results

The new features included in the media players (*SpinPlayer v2.2* and *streamplay*) and described in Section 3 were jointly validated by Spin Digital within the scope of the technical preparation of the final project demonstration. In order to address this testing and validation phase, an immersive and interactive media playback system for high-quality high-resolution 360° video and HOA sound was built in the Spin Digital lab. This playback system is depicted in the next figure.

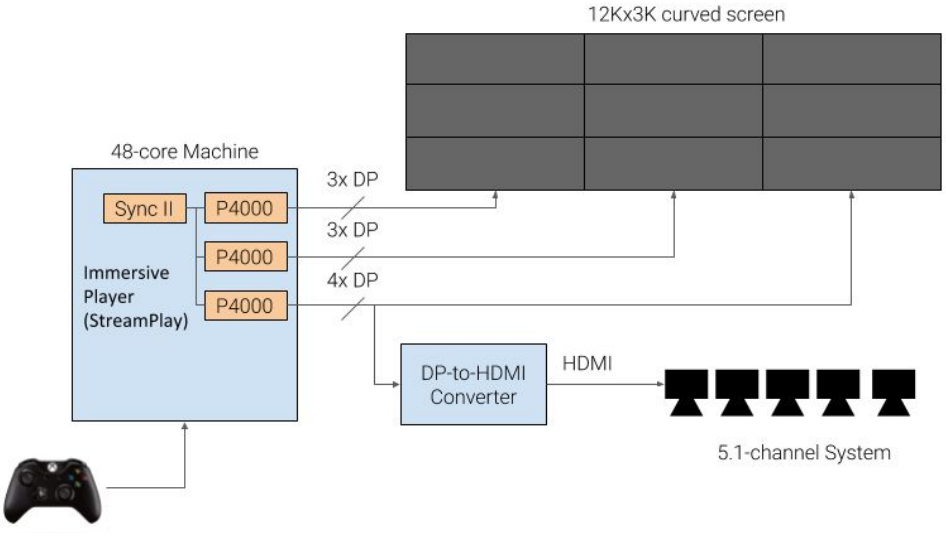


Figure 7: Playback system diagram

The playback machine included two Intel Xeon Platinum 8260 (2x24 cores) processors and three NVIDIA Quadro GPUs P4000. The GPUs were connected to the 12K curved screen layout using DisplayPort cables. The audio signal was sent out via DisplayPort, but afterwards converted to HDMI for feeding the 5.1-channel system.

The set of test sequences was the video and audio material that PSNC prepared for the final project demonstration, in particular:

- *The Noise and The Structure*: 14931x4247 pixels, ERP, 29.97 fps, 3rd-, 5th-, and 7th- order HOA and LFE
- *Krambabula 1*: 15152x7576 pixels, ERP, 25 fps, 3rd-, 5th-, and 7th- order HOA and LFE
- *Krambabula 2*: 15152x7576 pixels, ERP, 25 fps, 3rd-, 5th-, and 7th- order HOA and LFE
- *Fallout Shelter*: 12960x8640 pixels, CMP, 50 fps, 3rd-, 5th-, and 7th- order HOA and LFE

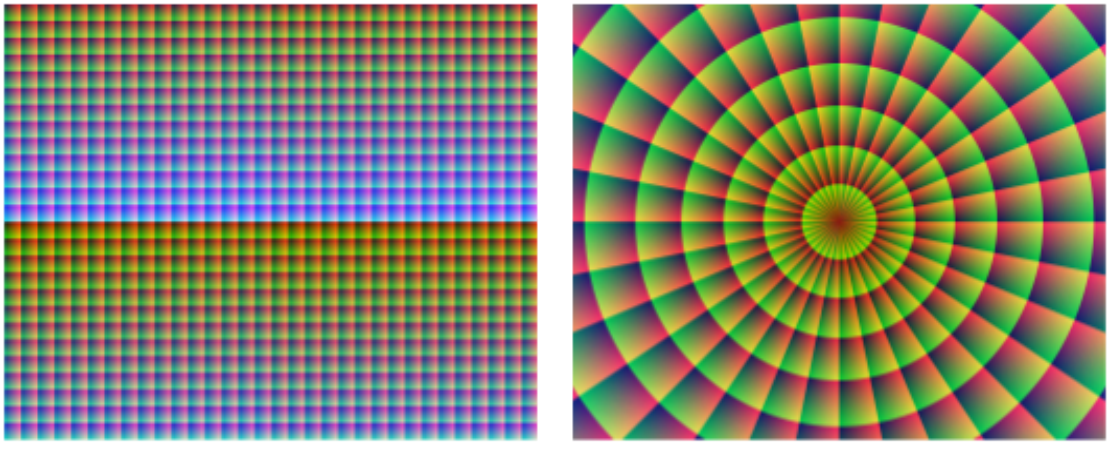
These sequences were encoded using the Spin Digital HEVC encoder at very high bitrate (200 Mbps for *The Noise and The Structure*, *Krambabula 1 and 2*, and 600 Mbps for *Fallout Shelter*). Before encoding, *Fallout Shelter* was converted from CMP to ERP with a resulting resolution of 14964x7482 pixels. The HOA and LFE files were encoded using the ffmpeg OPUS encoder at 128Kbps per channel and then encapsulated together with video using the MKV format.

Some informal tests were conducted to validate the new features of *SpinPlayer v2.2* and *streamplay*. Note that, since these media players also include the features that have been described in phases 1 and 2, some of those features (e.g, AVX 512, BC4 texture format, NUMA, decoupled 360° video rendering) were also used to make high-quality high-resolution 360° playback work.

These performed tests are described next:

Equirectangular projection (ERP)

The equirectangular projection was validated using a color blocks sequence created with the allrgb source filter from FFmpeg. The original video sequence can be seen in Figure 8a, where the video geometry format is set as plane. When a video frame like this is interpreted as 360° video, the vertical lines represent latitude in a sphere (a line connecting north and south poles), and horizontal lines represent longitude. In Figure 8b, the viewport is showing the north pole, where all vertical lines connect with themselves, and the horizontal lines become circles.



(a) with plane projection (b) with ERP and viewport at the pole

Figure 8: ColorBars sequence where the geometry is set as (a) plane and (b) ERP

Curved screen support with cylindrical projection

The cylindrical output format was validated in Spin Digital Lab with curved screens, and the color blocks video sequence. Here, the vertical lines were equally spaced, and they were physically measured.

Multi-device support with interactive 360° video & 360° navigation with the Xbox controller

The interactivity of 360° video with multi-device support was tested with three GPUs and nine curved monitors with mouse and Xbox controller. In both scenarios, the viewport changes were smooth, and according to what was expected. Navigation with a mouse was cumbersome, with several movements needed to change the viewport. On the other side, the Xbox controller was much more convenient and natural.

New licensing mode

The new license modes which do not require a dongle connected to the system were also validated. Herein, both the RMS Leased License mode and Sentinel Cloud Connected mode were tested in a simulated environment in Spin Digital Lab.

HOA sound extension

The support for custom layouts was tested during the development of this feature by checking that all the attributes included in the JSON were parsed correctly to the HOA decoding library. To this end, JSON files describing a 5.1-channel system with 3rd-, 5th-, and 7th-order decoders were created using the IEM *AllRADDecoder* plugin. These files also served us to test the new rotation and zoom algorithms for up to 7th order. *Streamplay* was executed to playback the video files with HOA sound with the custom layout option: `--isambix 1 --layout custom --json <file_name>.json`

The video and audio scenes were jointly rotated and zoomed in/out with the XBOX controller. We could verify visually that both elements were synchronized and reacted properly to the user commands. From the subjective point of view, we did not see any issues related to 5th- and 7th-order zooming and rotation using the 5.1-channel soundbar. It is also worth mentioning that the new rotation algorithm is still experimental, as further tests on a professional sound equipment, such as the 23.1 loudspeaker system at PSNC, are required.

Complete immersive media player validation

All the new features for immersive media playback including high-resolution 360° video, high-order ambisonics, and user interaction were jointly validated at the Spin Digital lab. The pictures below show a screenshot of the playback of *Fallout Shelter* and *Krambabula 1* in 15K original resolution with a 12K viewport resolution displayed on the curved screens. The input audio is 5th-order ambisonics reproduced on the 5.1-channel speaker system. User interaction (rotation and zoom) was performed using the Xbox controller. Geometric adjustment for curved screens is applied in real-time for a FoV of 120°.

Figure 9: Validation of immersive set-up at Spin Digital Lab

4.4 Specification of media playback systems

The Spin SDK has been extensively optimized for performance with advanced multithreading in order to make efficient use of modern multi-core CPU architectures. Hence, the performance of the Spin SDK and media players scale very well with the number of CPU cores. By using CPUs with higher core counts, it is possible to increase the performance for applications that require higher bitrates, higher quality chroma formats (4:2:2 or 4:4:4), higher frame rates, and even resolutions beyond 8K.

In Section 4 of the Deliverable 5.3 - Report on QA and Content Preparation Guidelines, examples of hardware settings for different use-cases envisioned in the Immersify project are presented. Additional recommended media playback systems will be presented along this document for the other media players in the next sections. Nevertheless, the recommended playback systems presented herein encompass the three integration phases reported in the previous section.

4.4.1 Integration Phase 1: 16K video playback

The target format supported by this integration phase is 16K resolution at 60 frames per second, with chroma subsampling 4:2:0, 10-bit depth, and at the maximum bitrate possible In this scenario, a high number of CPU cores is needed to cope with a large amount of data. Moreover, four GPUs are required to provide a 16K display signal, where they should be synchronized. In our tests, the NVIDIA GPUs have presented better synchronization performance than the AMD GPUs. In Table 6, the summary of the recommended platform and video specs for this integration phase is shown.

Table 6: Recommended media playback system for integration phase 1

Item	Description
Media Player	Spin Player
OS	Windows 10 64-bit
CPU	2x Intel Xeon Platinum 8260 (48 cores)
GPU	4x NVIDIA Quadro P4000 with NVIDIA Quadro Sync II board
Memory	96 GB (12x 8 GB, DDR4 2933)
Display	16x 4K monitors (or projectors)
Video specs	HEVC Main 10 profile: 4:2:0 - 10-bit
Max. video resolution, frame rate and bitrate	16K at 60 Hz with 400 Mbit/s

4.4.2 Integration Phase 2: Multi-device and responsive 360° playback

For responsive 360° playback of integration phase 2, interactivity in multi-device environments was not supported. Therefore, only one GPU is required. The recommended omnidirectional input video frame should have 12K (11520x4320) resolution at 60 frames per second. Moreover, the chroma subsampling format is 4:2:0 and 10-bit depth. The recommended platform summary is in Table 7.

Table 7: Recommended media playback system for integration phase 2

Item	Description
Media Player	Spin Player
OS	Windows 10 64-bit
CPU	Intel Core i9-10980XE Extreme Edition Processor (18 cores)
GPU	NVIDIA Quadro P4000
Memory	32 GB (4x 8 GB, DDR4 3200)

4.4.3 Integration Phase 3: Immersive and interactive 360° video with spatial audio

In the integration phase 3, interactivity with multi-device was added along with spatial audio. The suggested media player, in this case, is the streamplay, where the MOTU 24Ao device is recommended, which includes a USB audio interface with support for WASAPI. The maximum omnidirectional resolution supported is 16K. The video format is 16K at 60 fps, 400 Mbps, with 4:2:0 chroma subsampling as in the integration phase 1, but in this case it is a 360° video. However, the maximum output resolution is limited only by the GPU. Hence, the maximum number of NVIDIA Quadro boards with NVIDIA Quadro Sync II is recommended.

Further, the Xbox controller is selected as the interactivity interface. In order to immerse the user into the 360° video playback, curved displays in a cylinder shape are suggested. In this case, the SpinRender output projection format must be the cylinder. Table 8 presents a summary of the recommended settings.

Table 8: Recommended media playback system for integration phase 3

Item	Description
Media Player	streamplay
OS	Windows 10 64-bit
CPU	2x Intel Xeon Platinum 8260 (48 cores)
GPU	4x NVIDIA Quadro P4000 with NVIDIA Quadro Sync II board
Memory	96 GB (12x 8 GB, DDR4 2933)
Audio	MOTU 24Ao
Interactivity	Xbox controller
Display	Multiple curved displays (cylinder output projection)

4.5 Advances over the State of the Art

Herein, state-of-the-art media players were selected to be compared with the Spin Player and streamplay solutions. The criteria to choose the players are based on immersive video playback. Among those, a sample set of 4 players were selected, which provide support to:

1. The HEVC standard;
2. 360° video formats: cube map or equirectangular
3. Large displays
4. 360° scene navigation

The players selected are: Movies & TV from Microsoft, VLC from VideoLAN, 5KPlayer from DearMob and GOM Player from GOM & Company. The main capabilities of Spin Player and streamplay of integration phase 3 are compared to the four selected media players in Table 9.

Table 9: Comparison of Spin Player and streamplay against state-of-the-art media players

Spin Player	streamplay	Movies & TV[17]	VLC[18]	5KPlayer[19]	GOM Player[20]
Maximum input resolution					
>16K	>16K	8K	8K	8K	8K
Maximum output resolution					
16K	16K	16K	16K	8K	8K
Decoupled render					
✓	✓	✗	✗	✗	✓
Multi-device rendering					
✓	✓	✓(mosaic)	✓(mosaic)	✗	✗
Low-latency viewport updates					
✓	✓	✓(paused)	✗	✗	✓
360° video interactivity					
Xbox or mouse	Xbox or mouse	mouse	mouse	mouse	mouse
Curved displays support					
✓	✓	✗	✗	✗	✗
High Order Ambisonics					
✗	✓	✗	✗	✗	✗

Due to the software decoder optimizations of SpinDec, the SpinPlayer and streamplay can playback resolutions up to 16K with the right amount of CPU cores. On the other hand, players like Movies & TV and VLC are restricted by the hardware (GPU decoder) to maximum 8K video. The 5KPlayer and

[17] Microsoft Windows 10 default media player
[18] https://www.videolan.org/vlc/download-windows.html
[19] https://www.5kplayer.com/
[20]

GOM Player, although they do not use GPU hardware video decoders, are not able to playback files with a resolution higher than 8K.

Large display support via multi-device rendering mode from Spin Player and streamplay provides much better scalability of the multi-screen support for high-resolution videos compared to the selected media players. These media players use the multi-device support enabled by GPU drivers (Mosaic or Eyefinity). Therefore, the theoretical maximum output resolution supported by these players is limited by the GPU driver or Windows Desktop, which is 16K x 16K. Achieving 16K x 8K playback using GPU driver multi-device is, however, not trivial, as only particular combinations of GPU models, GPU driver, and OS version work well together to create a working system. Even if these players can work on such systems with a desktop resolution at 16K x 8K, they are still not able to do a 16K x 8K video playback at 60 fps. So far, the video player developed by Spin Digital is the only solution that supports 16Kp60 video playback.

The decoupled rendering mode, which grants low-latency changes in the viewport, is also not supported by all players. Among the tested, only GOM Player was able to provide low latency such as Spin Player and streamplay. Nevertheless, all tested players were able to navigate on omnidirectional videos with the mouse. Only Spin Player and streamplay were able to support a more convenient interaction interface such as the Xbox controller.

Finally, none of the tested players could provide more immersive features, such as the curved display support via cylinder output projection and high-order Ambisonics sound.

5 Native Core Plugin and Immersify Video Player

5.1 Concept

In the context of the Immersify project, we implemented several players for the Deep Space 8K using Spin Digital technology for decoding. We made a clear distinction between rendering code and decoding code in our player implementations for Deep Space 8K. As a result, the decoding part of our code could be shared between several players running in three different environments (OpenGL, Unity3d, Unreal engine). The players are built on the top of a library called “Native Core Plugin”. Native Core Plugin is a C++ library that serves as an interface to the Spin Digital HEVC Decoder library named SpinDec. SpinDec runs in the background and decodes video bitstream that has been encoded accordingly with the HEVC video standard.

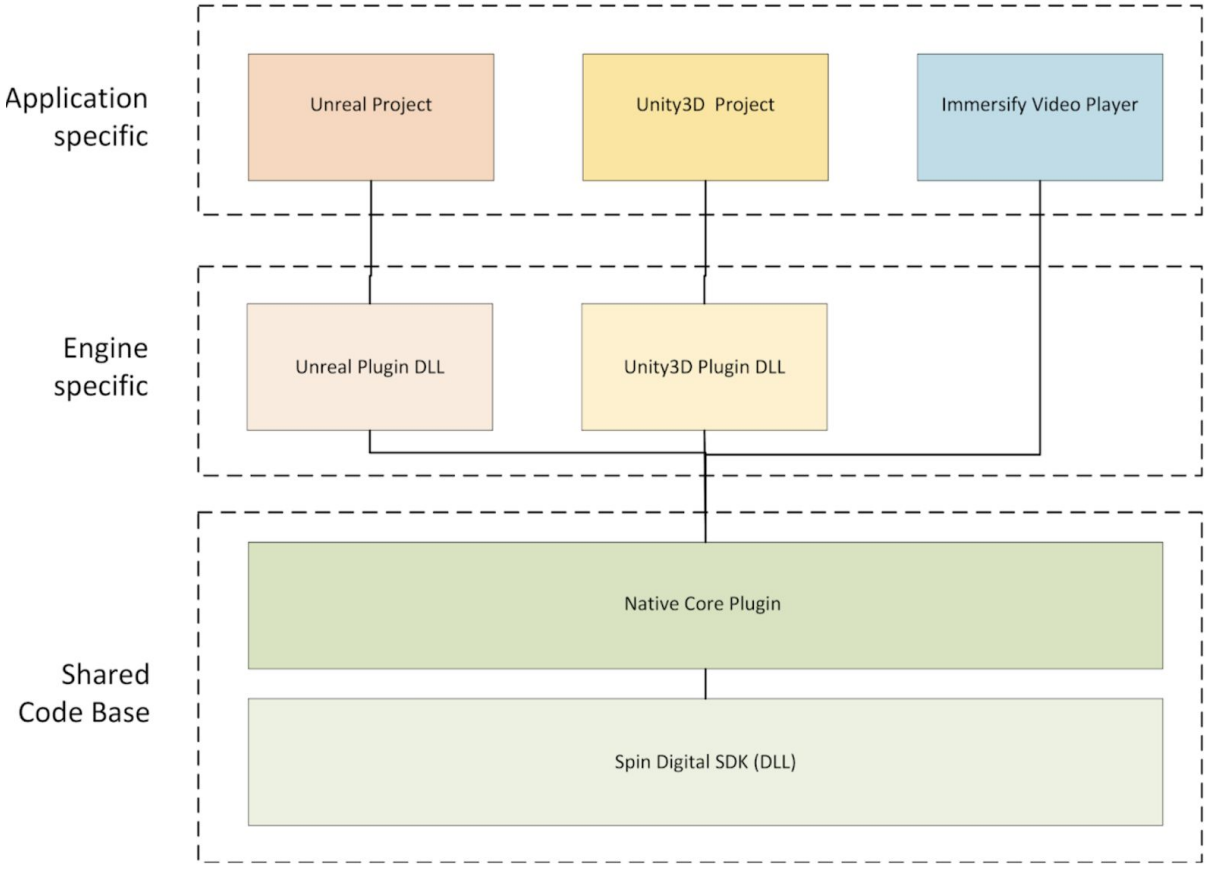


Figure 10: The software architecture of three Immersify Players implemented in Unreal, Unity3D and OpenGL and their underlying Native Core Plugin

5.2 Implementation

5.2.1 Native Core Plugin

Native Core Plugin can be accessed in any standard C++ application by including the “sequencer.h” interface.
The sequencer component serves as an interface to the plugin and implements several methods for uploading, decoding and playback of video files. The following section shows an overview about the most important methods of the sequencer:

- **void setTextureAccess**(BaseTextureAccess* textureAccess): This method has to be called before other methods. It links a texture, in which the video data will be written, to the sequencer. It takes an instance of the BaseTextureAccess as an input. BaseTextureAccess is a base class that can be derived by either OpenGL-texture or a Directx-texture (whereas GLTextureAccess is supporting OpenGL-Technologie and DXTextureAccess uses Direct-X).
  Obviously, in the case of (OpenGL) Immersify Video Player, only the GLTextureAccess class is used.
  GLTextureAccess needs to be set up with three textures (Y, U, V). These textures will be filled with the data during the runtime. Therefore, they have to be initialized according to the size of the video. Please note that these textures might have different sizes depending on the chroma subsampling settings (4:4:4, 4:2:2, 4:2:0), the width and height of the video and the texture format. In order to increase the uploading speed of texture data to the GPU, we use BC4 compression, which is a compressed texture format commonly used in DirectX and OpenGL applications (as GL_COMPRESSED_RED_RGTC1).

- **void setDecoderOptions**(int numPictureBuffer, int decoderNumThreads, int maxQueueSize, bool writeLogs=false). This method sets the default options for the decoding process. The input values can be set by the users and will be described in more detail later in this document.

Please note that the sequencer class expects a specific order of calling methods. The two methods mentioned above have to be executed before any further calling of the sequencer methods.

- **void play**(float framerate=60.0f, bool shouldLoop=false): starts the playback using a predefined framerate. The *shouldLoop* parameter can be enabled to make the video loop at the end. In case the application was started with logging enabled, the *shouldLog parameter* will be true and the plugin generates a text file where some information about the actual performance will be written (can be useful for tests and analyzing purposes).

- **void setPause**(bool pause) sets the current state of playback to paused or unpaused.

- **bool update()** this method should be called in fixed intervals to update the texture. Please note that the update interval must be either shorter or equal to the update interval of the corresponding framerate.

- **void stop()** stops the video.

- **bool isPaused()** returns the paused/unpause state of the video.

- **bool isFinished()** returns true if the video has played to the end and is finished, otherwise false

- **int getCurrentPlayingTime**(): returns the current playback time in milliseconds. This method is also useful for synchronization. Using this method, a master instance of Immersify-OpenGL player could estimate the current playback time and send it to the slave instance of the application. The slave instance then calls *setTargetPlayingTime* to synchronize the playback time with the master instance.

- **void setTargetPlayingTime**(float targetPlayingTime) sets the desired target playback time. This value should be close to the actual playback time. *setTargetPlayingTime* is mainly used for synchronization (that includes synchronization between audio and video as well as synchronization between different instances of the video player (e.g., on different machines).

- **void overrideTargetFPS**(float fps) overwrites the frame rate while playback (this method could be used for speeding up or slowing down the playback speed).

- **bool isReady**(): returns whether the sequencer instance is ready to start the playback. It returns true when all required software instances are initialized and the video file is loaded.

- **PLAYER_STATE getPlayerState**(): returns an enum descripting the current player state. Defined values are: STOPPED, PAUSED, PLAYING.

- **VideoInformation loadMP4**(const char* src_filename): returns selected meta-information about the video file. It includes the resolution, the framerate and the chroma-subsampling level of the video stream.

5.2.2 Immersify Video Player

In this section, we will focus on the executable component (Immersify Video Player) and describe its capabilities regarding video and audio playback as well as network and playback synchronization. The Immersify Video Player is an OpenGL-based video player written in C++ for Deep Space 8K at the Ars Electronica. It implements an interface to Spin Digital SDK, which is running in the background for decoding.

Figure 11: Immersify Video Player playing a high resolution stereoscopic video (Deep Space 8K, wall and floor projection)

1) Video Support

Immersify Video Player supports HEVC video bitstreams. Other video formats, like AV1, are not supported. Furthermore, the player supports 4:2:0, 4:2:2 and 4:4:4 chroma-subsampling and can also playback side-by-side stereoscopic videos. Besides, multiple instances of the Immersify Video Player can be synchronized and be running on distributed machines that build together an extended display.

2) Audio Support

Immersify Video Player uses Microsoft MCI player in the background for audio playback. Windows Control Interface (MCI) is a Windows interface (supported in all modern Windows distributions) and provides standard commands for playing media content. It offers a generic interface and provides applications with the ability to playback audio over all audio devices that are installed on the host machine. As it uses Microsoft MCI player, it can play every audio format that is supported on the host machine, including multi-channel audio and compressed audio formats like MP3 or Ogg.

3) Synchronization

The player supports synchronized video playback over multiple instances. The synchronization is based on the playback time. A hardware-based synchronisation (e.g. via swap barriers) is not included in our implementation since it is not supported in all our target engines. In our implementation player instances are defined either as master or slave. In a typical use-case for Deep Space 8K we use a master instance for the wall projection along

with a slave instance for the floor projection. The master instance will continuously be sending information about the current playback time to the slave instance and the slave instance will be receiving this information and synchronizing the playback time with the master instance. The synchronization is implemented through a messaging system that uses the network for communication. The following subsection provides further details about network communication.

4) Network Commands

Instances of Immersify Video Player can be controlled via UDP commands coming from the network. These commands can be used for starting, stopping, pausing, and quitting the application as well as for playback synchronization. The IP and the port for the slave instance can be defined via command-line arguments for master instances. For slave instances, only the port has to be set (via command line). Below all supported UDP commands are listed.

- **AT** (Application Terminate): this command closes the application.
- **AP** (Application pause): this command should be used to pause the playback.
- **AR** (Application Resume): continue the playback (should be called when the video is paused).
- **TM{ms}** (Time Message): defines the current playback time in milliseconds. For instance, the message "TM10520" means the desired playback time should be 10520 ms now.

5) Command-Line Options

Applications in Deep Space 8K are handled in a fully automated way. Therefore, applications usually do not include a graphical user interface. Instead, we use command-line arguments to run the player with corresponding parameters. Below all supported arguments for Immersify Video Player are listed:

-filename: the path to the video, e.g., C:/path/to/the/video.hevc (it can be a relative or absolute path).
-**framerate**: the frame rate of the video. On default, the frame rate will be taken from the input file.
-**loop**: defines whether the video should be looped.
-**start**: by default, the video player will pause after initialization and will be waiting for the start-command coming from the network. By using the start argument, the player will start immediately after the video file has been loaded and initialized.
-stereo: defines whether the input video should be rendered as a (side-by-side) stereo video (otherwise the video will be rendered as a mono video) .
-**audiofile**: the path to the audio file. The playback of the audio will start together with the video playback.
-threads: defines the number of threads that shall be used for decoding. Default is -1 and uses the available resources as best as possible.
-**ifpics** (in fly pictures): defines in SpinDec the number of concurrent pictures that shall be decoded in parallel (this value corresponds to "iNumConcurrentPics" in SpinDec). The default is -1, where the library makes this decision.
-**maxqueuesize**: defines the maximum number of frames that shall be decoded in advance.

The default value is -1 and takes the default value from the library (16 buffered frames will be pre decoded). The maximum number depends on the hardware / RAM capacity.
-log: if set, log files will be created. It can be useful to evaluate the performance of the player and decoder.
-fps: if set, current frame durations will be shown on the upper right corner of the application. This option can be useful for performance testing.
-master: If set, the application instance will run as master and will be sending the current playback time continuously to the slave instance.
-slave: If this option is enabled, the application will run as slave and will be receiving current playing time messages
-slaveIP: defines the IP of the slave instance.
-si: defines the swap interval of the application.
Swap interval is a commonly used method to prevent frame "tearing" which occurs when a half of a frame is shown with a half of the previous frame. It can be very annoying especially for high motion video contents. Swap interval synchronizes the exchange between the front and rear frame by using a hardware event called vertical blanks (v-blank).
-noasync: no audio synchronization. If this option is enabled, the application will not try to synchronize the playback of the audio file to the video playback, although they will start together.
-syncDelay: for perfect synchronization, sometimes it is necessary to consider the time it requires for a message to be delivered. The syncDelay option adds a delay in milliseconds to the current playing time, so that the slave instances will have the actual playing time of the master instance on the time of receiving the message. This argument has only an effect if the master option is set.
-udpUpdateInterval: defines the time interval between synchronization messages in milliseconds (valid only for the master instance). For example, if it is set to 50, the master instance will send a synchronization message to the slave instance every 50 milliseconds.
-glutinit: Using glut game-mode could be beneficial for the sake of performance improvement. The value of this parameter is a string that will be directly used for initializing the glut: void glutInit(int *argcp, char **argv)

5.3 Results

The output of this implementation contains two components: a C++ library and an executable. The C++ library (Native Core Plugin) serves as a video playback library that interfaces SpinDec in the background. It can be linked to different applications and be used in different software environments like Unity3D, Unreal Engine or a pure OpenGL application. The second component is a high-resolution video playback application that is based on OpenGL technology and can be used on any machine running Windows 7 or later. The main challenge for the implementation of the rendering pipeline in OpenGL was the asynchronous loading of textures to the GPU. In general, loading a high-resolution texture onto the GPU is a costly process. This can be a critical factor for playback performance, depending on the hardware and the resolution of the video. We solved this problem by using an OpenGL extension called PBO (Pixel Buffer Object) which allows us to transfer pixel data asynchronously to the GPU. Besides, we used BC4 compressed texture format to reduce the required PCI bandwidth.

5.4 Specification of media playback systems

The system specification required for the Native Core Plugin and the corresponding Immersify Video Player is very similar to the specification of the Spin SDK as it is used in the background for decoding. Spin SDK uses advanced multithreading technology of modern multi-core CPU architectures. Therefore, systems with more CPU cores are generally expected to perform better. Additionally, for Immersify Video Player, stereoscopic video playback requires systems that support quad-buffer stereo. We have tested our software on various hardware systems. Although a hardware requirement for video playback is highly dependent on the video resolution, frame rate, and chroma subsampling of the video, we have successfully tested high-resolution videos (e.g. 11K x 11K @30) with the following specification: Windows 7 SP1 64 Bit, 2x Intel Xeon CPU ES-2687W (2x8 cores), 4x NVIDIA Quadro P6000, and 64 GB of RAM. However, for videos with higher resolution (e.g. 12K x 12K @30 and beyond) it was necessary to use a system with more CPU cores: Windows 10 Pro (v. 1909), 2 x Intel Xeon Gold 6140 (2x18 cores), 2 x NVIDIA Quadro P6000, and 96 GB of RAM.

5.5 Advances over the State of the Art

This section will discuss the advances of Native Core Plugin and Immersify Video Player over existing technologies. The ability to playback super high resolution video content is essential for immersive contents like 3D or 360°. Deep Space 8K player combines two different fields of technology: in the background it uses Spin-Decoder for decoding and OpenGL for rendering. To our knowledge there is no OpenGL, Unreal or Unity3d implementation that supports video playback at the same level of resolution as our player (up to 16K). Specially for 3D-engines ultra high resolution video playback can be a very useful feature in order to playback 360° mono/stereo video content with sufficient resolution and mix it with embedded virtual objects. Beside, the projection area of our OpenGL-based Immersify Video Player can be extended by using multiple instances in order to generate even larger projection areas.

6 Unity Immersify Player

6.1 Concept

Thanks to a native C++ plugin for Unity3D, the so-called “Unity Immersify Player Plugin”, it is possible to provide the Spin Digital HEVC decoder functionality to Unity3D. Therefore, we integrated the component, that is described in Section 5, Native Core Plugin and Immersify Video Player, into the Unity Immersify Player Plugin.

The Unity3D plugin wraps the Spin Digital HEVC decoder functionality and provides an interface for Unity3D to load, decode and buffer an HEVC video bitstream (raw or within an MP4 container). The Unity3D plugin reads the video file with the help of the Spin SDK, caches the pre-loaded frames in a buffer and draws the current frame to a texture that has been created within Unity. Therefore, the texture containing the video frame can be used like any texture within Unity. Usage examples are provided to show planar and 360° videos in mono and stereo. The equirectangular and equiangular texture-format can be used to show 360° videos. The plugin supports the Windows desktop platform (x64) and can be used with OpenGL and DirectX11 together with the Unity XR features.

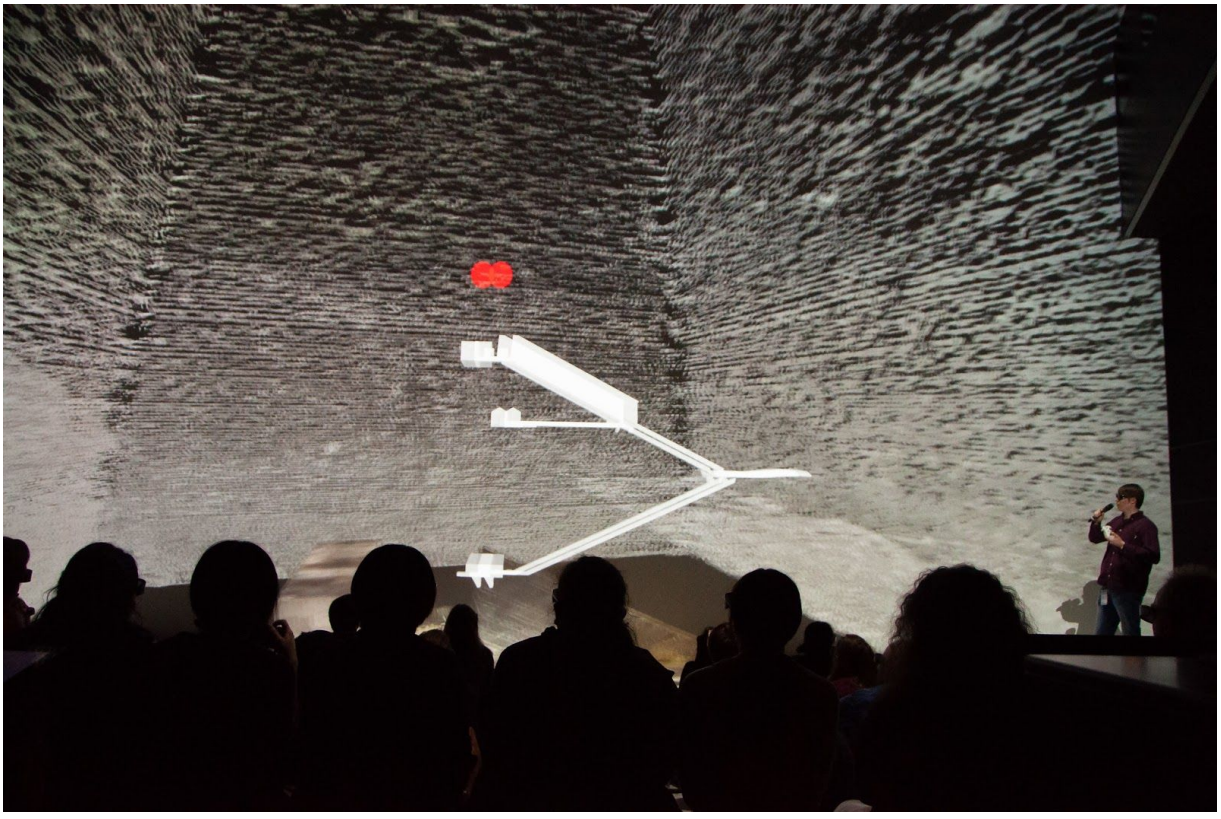


Figure 12: Embedding virtual objects in high resolution stereoscopic 360° video content using Immersify Unity3d Plugin (Deep Space 8K, wall and floor projection)

In this section, we are going to get an overview of all components that were created to implement the Unity Immersify Player, including the API that was implemented to use the native C++ plugin.

6.1.1 Core Components

The core components / classes to use the plugin are:

- **ImmersifyPlugin.cs**: used to load and play an HEVC bitstream. This class is the interface to the native plugin.
- **ImmersifyMeshVideo.cs**: the glue between the ImmersifyPlugin and the mesh that displays the video texture.
- **VideoUvController.cs & EquirectangularUvController.cs**: takes care of the video uv-settings for mono & stereo videos. The first one is used for planar, the other one for 360° videos.
- **LoadVideoByConfig.cs**: Helps to load a video, that is specified in the configuration or via start parameters at startup.
- **ImmersifyCmdConfigMgr.cs**: The "Immersify Command Config Manager" makes it possible to define default settings in the editor and to overwrite these settings as command line arguments when starting a build.

6.1.2 Shaders

Various shaders are used to display the video (BC4 compressed) within a Unity scene:

- **ImmersifyPlaneBC4.shader**: This shader is used to render the BC4 compressed textures to a plane (or any other) mesh. Use case: Render an e.g. 16:9 video to a plane in space.
- **ImmersifyEquirectCubeBC4.shader**: This shader is used to render an equi-rectangular format (2:1) video to a unity standard cube mesh. This shader renders to the background. Use case: Render a 360° equi-rectangular video.
- **ImmersifyUnlitTextureBC4.shader**: This shader is used to render an equi-angular video to a cube that was specifically created for this use case. The main difference to the ImmersifyPlaneBC4 shader is that it renders to the background. Use case: Render a 360° equi-angular video.

6.1.3 Scenes

Several Unity scenes are provided to show different use cases how the plugin can be used:

- **ImmersifyEquiAngularVideo**: This is a 360° video player scene for equi-angular videos[21]
- **ImmersifyEquiRectangularVideo**: This is a 360° video player scene for equi-rectangular videos[22]
- **ImmersifyHmdVideo**: This is a scene to display 360° videos for HMDs like Oculus Rift or HTC Vive. For demonstration purposes, the Oculus SDK and camera rig was chosen, but other HMDs may be implemented in a similar way.
- **ImmersifySinglePlaneVideo**: This is a planar video player scene with a plane that can be put into the world to play the video.

[21] An introduction to Equi-Angular Cubemaps:
https://blog.google/products/google-ar-vr/bringing-pixels-front-and-center-vr-video/
[22] An introduction to Equi-Rectangular Cubemaps:
https://medium.com/@onix_systems/how-to-use-360-equirectangular-panoramas-for-greater-realism-in-games-55fadb0547da

- **ImmersifyMultiPlaneVideo**: This is the same as the single plane video scene, but contains two videos to show that multiple videos can be played.

6.2 Implementation

The implementation of the Unity player is actually an implementation of the Core Player Plugin interface with additional scripts to display the video within a Unity scene. The interface, that is implemented within ImmersifyPlugin.cs, contains the following functions:

- IntPtr **CreateSequencer**(string path, int numberOfThreads, int videoBufferSize, int maxQueue, bool shouldLog);

  This creates a new sequencer that loads the video, that can be found at the specified path, and returns a pointer to the player instance, which is used as reference to do all further operation with the video. Parameters are the path, where the video file exists on the harddrive, the number of threads that shall be used for decoding, the amount of frames that shall be buffered and the maximum queue (maximum buffered frames). The *shouldLog* parameter can be used if you want the C++ plugin to log some information to a txt file (interesting for performance tests and analyses).

- void **InitPlayer**(IntPtr playerInstance, IntPtr texturePtrY, IntPtr texturePtrU, IntPtr texturePtrV, int format);

  This is the entry point for creating a player instance. It needs to be setup with 3 textures (Y, U, V) (take care, that these textures might have different sizes depending on the chroma subsampling settings, 4:4:4, 4:2:2, 4:2:0) and the texture format (casted to int, as reference please have a look at UnityEngine.TextureFormat[23]).

- void **Play**(IntPtr playerInstance, float framerate, bool shouldLoop);

  This function needs to be called to play the video at a specified framerate. The *shouldLoop* parameter can be set to true to make the video loop at the end.

- bool **IsReady**(IntPtr playerInstance);

  This function can be called after *InitPlayer* and before *Play* to check if the player is ready to start playing.

- bool **IsFinished**(IntPtr playerInstance);

  This function can be called to check if the video is finished (last frame got played). This call only makes sense for non-looping videos.

- void **SetPause**(IntPtr playerInstance, bool pause);

  Call this and set the pause parameter to true or false to pause or unpause the video.

- bool **IsPaused**(IntPtr playerInstance);

  This function returns a bool that tells if the video is currently paused (true) or unpaused / running (false).

[23] UnityEngine.TextureFormat: https://docs.unity3d.com/ScriptReference/TextureFormat.html

- void **PauseAfterFirstFrame**(IntPtr playerInstance, bool pauseAfFrame);

  This can be used (call before calling play) to make the video pause after displaying the first frame. By default, the video starts to run as soon as the first frames are loaded. This function can be used to sync the start of a video between multiple instances.

- void **OverrideTargetPlayingTime**(IntPtr playerInstance, float targetTime);

  This overrides the current target playing time (and implicitly adopts the target FPS). Instead of moving the clock on with a given FPS amount, the time that shall be played "now" is set with *targetTime*. This is used for synchronously played videos, where the master tells the slave, where the video is supposed to be.
  Use this method, if you have a target time and want the video to be at this exact time. A use case might be to synchronize this video with an external time-code. Calling this overwrites a previously set target frame rate with the default video frame rate.

- float **GetCurrentPlayingTime**(IntPtr playerInstance);

  This function returns a float that represents the current playback time in ms. (E.g. 1000 means that playback is at 1st second.)

- void **OverrideTargetFPS**(IntPtr playerInstance, float fps);

  This overrides the current target FPS (and implicitly adopts the target time). Instead of showing a frame that corresponds with the time that is set from outside, the plugin uses an internal clock driven by the defined frame rate to know when to show the next frame.
  Call this, if you want to play the video with a constant frame rate. Calling this overwrites a previously set target time with the time, that is calculated by the frame rate over time. A use case is to slow down or speed up a video over time.

- float **GetTargetFPS**(IntPtr playerInstance);

  This function returns a float that represents the frame rate that is currently used. In case of synchronization, this value is calculated continuously and therefore might not be the framerate that was defined by the user, but a slightly faster or slower frame rate to sync with the master.

- int **GetVideoWidth**(IntPtr playerInstance);

  This function returns an integer that represents the currently loaded video's width in pixels.

- int **GetVideoHeight**(IntPtr playerInstance);

  This function returns an integer that represents the currently loaded video's height in pixels.

- int **GetVideoChromaSubsampling**(IntPtr playerInstance);

  This function returns an integer, that represents the currently loaded video's chroma subsampling. (0 = 4:2:0, 1 = 4:2:2, 2 = 4:4:4)

- int **GetMaxQueueSize**(IntPtr playerInstance);

  This function returns an integer, that represents the maximum queue size of the currently loaded video.

- IntPtr **GetRenderEventFunc**();

  This function is implemented to fulfill the Unity plugin requirements and add the plugin instance to the command buffer to be executed.

- void **UnityPluginUnload**();

  This function should be called when finishing the application to unload the plugin.

- void **RegisterDebugCallback**(debugCallback cb);

  This function registers a callback method that enables the C++ plugin to log via the Unity Editor interface. We made the experience that Unity runs unstable when using this feature. It can be used for debugging though.

6.3 Using the Unity Immersify Video Plugin

The following sections describe the core components and how to use them.

6.3.1 Immersify Plugin

The ImmersifyPlugin class can be set up in the editor to play a video that was encoded with the Spin Digital HEVC codec.

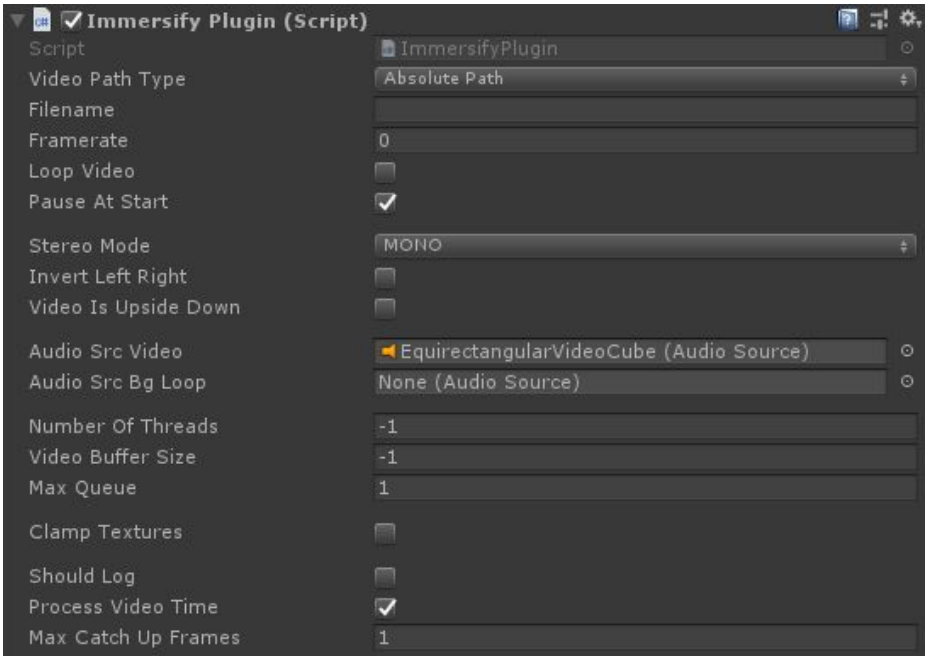


Figure 13: Screenshot of the ImmersifyPlugin script in Unity's Inspector view

The following parameters can be defined. A user always has to make sure that the values are correct for the video that shall be played. Wrong values will lead to a miss-behaving video or an application crash in the worst case.

Video Path Type: Indicates whether the following path to the video is an absolute system path or a relative path seen from the StreamingAssets directory.
Filename: The path to the video, e.g. C:/path/to/the/video.hevc
Framerate: The videos playback frame rate.

Loop Video: If true, the video is played again automatically after reaching its end. If false, the video is only played once.
Pause At Start: If true, the video pauses after it finished loaded and needs to be started manually. If false, the video starts playing automatically once loading is finished.

Stereo Mode: Set the stereo mode of the video: Mono (not stereo), Top-Bottom stereo or Left-Right stereo.
Invert Left Right: This is false by default. For a stereo video, the player assumes that the left or top part of the video contains the content for the left eye. If the left or top part of the video contains the right eye information, this needs to be set true.
Video Is Upside Down: If true, it flips the texture vertically by multiplying the y-axis in the shader by -1.

Audio Src Video: The audio source component can be linked optionally and will be played when the video is running (the video sound).
Audio Src Bg Loop: The audio source background loop can be linked optionally with an AudioSource component and is played when the video is paused.

Number Of Threads: Defines the number of threads that shall be used for decoding. Default is -1 and uses the available resources as best as possible.
Video Buffer Size: Defines the amount of concurrent pictures, that shall be decoded in parallel. (This value corresponds to “iNumConcurrentPics” in SpinDecoder. Default is -1 and lets the plugin (actually the SpinDigital decoder) make this decision.
Max Queue: Defines the maximum amount of frames that shall be decoded in advance. Default is -1 and lets the plugin make this decision (16 buffered frames will be saved). The maximum value depends on the hardware / RAM.

Clamp Textures: Required for some meshes, to set the render textures to *clamp* mode instead of *repeat* mode, to prevent visual artefacts.

Should Log: If true, log files are created that can be used to evaluate the performance of the player / decoder.
Process Video Time: If this is set to true (recommended), the Unity application tries to process the video with the ongoing time. In case of a lag, it tries to jump over video frames to catch up with the current time. If this is set to false, no corrections will be made which is correct in general but might not look so smooth sometimes.
Max Catch Up Frames: If the Process Video Time is true, the application never leaves out more frames than specified here. (A low number is recommended).

If the video contains sound, it is recommended to attach one or two Audio Sources to the GameObject, that hosts the ImmersifyPlugin, depending if there is only a normal video audio or an additional sound that shall be played while the video is paused.

6.3.2 Immersify Mesh Video

The ImmersifyMeshVideo class is the binding between the ImmersifyPlugin, that contains and manages the video texture, and the mesh, that displays the texture.

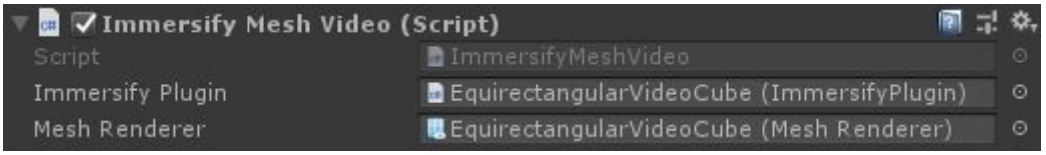


Figure 14: Screenshot of the ImmersifyMeshVideo Script in Unity's Inspector view

It requires a reference to the Immersify Plugin and the mesh that shall display the video.

6.3.3 Video UV Controller

The VideoUvController takes care about the correct display of a video. Especially for stereo videos, it handles the correct UV setting, to show the left video side when the left eye is rendered and vice versa for the right eye. If a stereo video is played in mono (e.g. because the video card does not support stereo), this component shows the left-eye side of the video.

Figure 15: Screenshot of the VideoUVController script in Unity's Inspector view

The VideoUvController component needs to be attached to a camera, because it requires the OnPreRender callback. If two cameras are used (separated eyes), the component needs to be attached to both cameras. If there are multiple videos played, all ImmfersifyMeshVideo components need to be referenced in the Immersify Mesh Video List.

The UvController needs to know about the ImmersifyMeshVideo component. (It gathers several information from this binding class.) During rendering, the UvController changes the UV transformation, so that mono and stereo videos are displayed correctly. It thereby respects the stereo related settings that have been done at the ImmersifyPlugin.

6.3.4 Immersify Cmd Config Mgr

The ImmersifyCmdConfigMgr (spoken "Immersify Command Config Manager") class is an extended version of the Command Config Manager from the Deep Space Dev Kit[24].

This class makes it possible to define default settings in the editor and to overwrite these settings as command line arguments ("starting parameters") when starting a build.

[24] The Deep Space Dev Kit: https://immersify.eu/home/guidelines-reports/deep-space-development-kit/

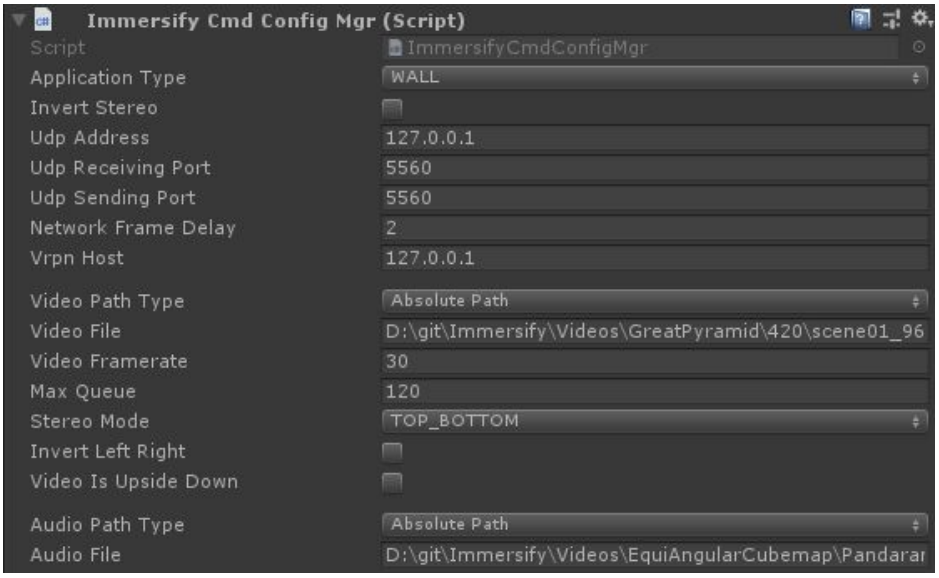


Figure 16: Screenshot of the ImmersifyCmdConfigMgr script in Unity's Inspector view

Beside other settings, it contains a subset of information to play a video. All parameters reference equally named parameters from the ImmersifyPlugin class. An exception to this is the way audio information is passed to the player. The Immersify Plugin holds a reference to an Audio Source, the ImmersifyCmdConfigMgr loads an audio file from a defined path (can be an absolute path or a relative one from StreamingAssets) and links the loaded audio clip to the the players audio source.

6.3.5 Load Video by Config

The LoadVideoByConfig class is a helping construct that is used to load a video specified via configuration at application start. The configuration is provided by the Immersify Cmd Config Mgr and is therefore consisting of default values or defined by starting parameters.

Figure 17: Screenshot of the LoadVideoByConfig script in Unity's Inspector view

If a video shall not be embedded in an application, but the Unity scene is only needed to play a specified video (as a normal video player does), this class is the easiest way to do so. It requires an ImmersifyCmdConfigMgr instance to be in the scene.

6.4 Results

The result of this project is a high end video plugin for Unity3D that allows users to play videos with very high resolutions (up to 16K x 16K) with up to 60 frames per second in mono and stereo (with active stereo glasses or head mounted displays) in the form of planar or 360° videos.

6.5 Specification of media playback systems

The system specifications required for the Unity Immersify Player are identical with the one from Section 5.4 (Specification of media playback systems), as this does the decoding in the background. Generally spoken, systems with more CPU cores are performing better. In our test with the Deep Space 8K (Windows 7 SP1 64 Bit, 2x Intel XEON CPU ES-2687W (2x8 cores), 4x NVIDIA Quadro P6000, 64 GB of RAM) we were able to playback 11K x 11K, 4:2:0 chroma subsampling, stereo videos with 30 FPS.

6.6 Advances over the State of the Art

In comparison with the Unity Video Player component[25], that comes with the engine, or Video Player Plugins, that are available in the Unity Asset Store like AV Pro[26], the Immersify Video Player is capable of playing videos with higher resolutions than possible with other players, and offers a generally better performance which results in a fast and stable frame rate. The table below shows this advance by comparing several features of mentioned players. The note "on supported hardware" in the comparison table shall make it clear that a strong CPU is required to play high resolution videos with 30Hz or 60Hz fluidly. The note "requires additional programming" means that this feature cannot be used out of the box, but it can be implemented on the base of this player.

Table 10: A comparison between the Standard Unity Video Player, the AV Pro Player and the Immersify Player focussed on Immersify related content

Comparison	Standard Unity Video Player	AV Pro Player	Immersify Player
1080p / Full HD	✓	✓	✓
4K / UHD	✓ (slow)	✓	✓
8K	✗	✓ (on supported hardware)	✓ (on supported hardware)
16K	✗	✗	✓ (on supported hardware)
VR / Stereo Support	✓ (requires additional programming)	✓	✓
360° Content	✓	✓	✓

[25] The Unity Video Player component: https://docs.unity3d.com/Manual/class-VideoPlayer.html
[26] The AV Pro Video Player: https://assetstore.unity.com/packages/tools/video/avpro-video-windows-57969

7 Unreal Immersify Player

7.1 Concept

We made the functionalities in Immersify Core Player (described in Immersify Video Player and Core Player component) also available in Unreal Engine 4 by creating a native C++ plugin. The provided usage example scenes also utilise Unreal's Blueprints visual scripting system to facilitate direct end-user interaction. We called the Immersify Player for Unreal "IMMPlayer" and the corresponding plugin that provides the player functionality, "IMMPlugin".

The Unreal Engine 4 Immersify Player plugin wraps the Spin Digital Player functionality and provides an interface for Unreal designers to load, decode and buffer a video file in the raw Spin Digital HEVC codec format (support for container-based formats like mp4 will follow in the future). The plugin decodes the video file frame by frame, caches the pre-loaded frames in a buffer and draws the current frame to a RenderTarget (a subclass of UTexture which is instanced at runtime). The RenderTarget is bound as a texture resource to an Unreal Dynamic Material Instance, which is also instanced at runtime. It is possible to create a Material within the Unreal Material Editor and assign it to be used by the plugin as a template for the instancing of the Video Material. Therefore, the resulting video material (containing the playing video content) can be used like any other Dynamic Material Instance within Unreal. Usage examples are provided to show planar and 360° videos in mono and stereo. The equirectangular and equiangular video-frame formats can be used to show 360° videos. The plugin supports the Windows desktop platform (x64) and can be used with DirectX11.

7.2 Using the Unreal Immersify Player Plugin

The following sections describe the end-user (developer that works with the IMMPlayer) components. It shows how to set up the Immersify Video Player in the Unreal Editor and provides an example on how to use the player.

At the core of the IMMPlayer is a C++ Unreal Plugin, which we call IMMPlugin, and which handles video file loading and decoding. Handling of advanced stereo and 360° formats is achieved by additional custom Unreal Material Shaders and Material Functions. Additional custom meshes address the needs for Equirectangular and Equiangular projections. Therefore, a virtual camera is placed in the center of this custom mesh to achieve the 360° effect.

The ImmersifyPlugin class can be set up in the editor to play a video that was encoded with the Spin Digital HEVC codec.

Preparation

For either a new or an existing Unreal Engine Project, the first step is to enable the plugin in Project Settings. You can find these settings in Settings TAB >> Project Settings >> Plugins.

Find the **IMMPlugin** under the **Immersify** label and tick on the 'Enabled' checkbox, if it is not ticked on already.

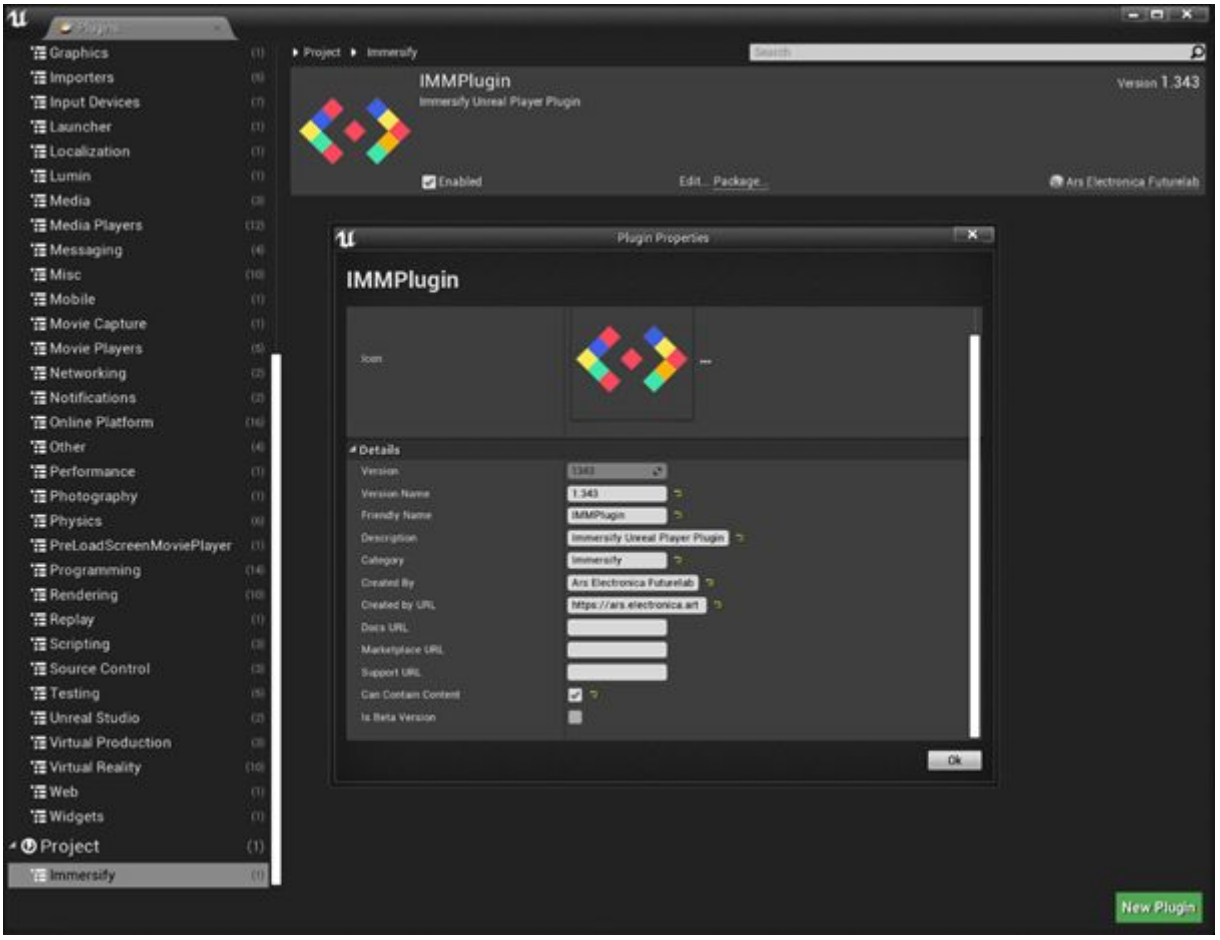


Figure 18: Screenshot of the Immersify Plugin properties in UE4

Note: A restart of the Editor is usually needed for any such change to take effect, this is standard procedure.

Scene Setup

Locate the **IMMSequencer** class in the Content Browser (IMMPlugin C++ Classes >> IMMPlugin >> Classes) window and drag-and-drop the icon into your level, which will immediately create an instance. If you cannot see the “IMMPlugin C++ Classes” directory, you need to enable “Show Plugin Content” in the Content Browsers view options.

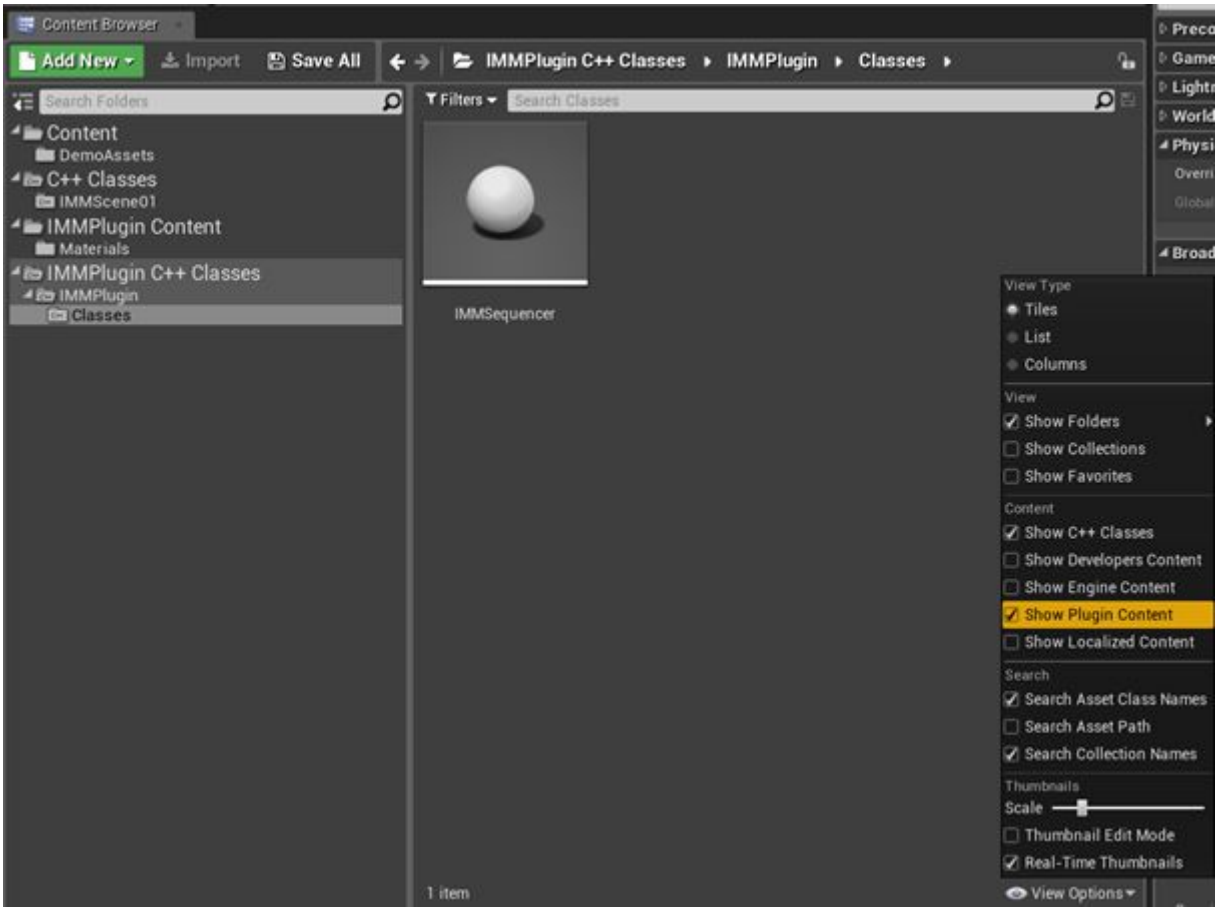


Figure 19: Screenshot in UE4 to enable “Show Plugin Content” to be available to see the IMMSequencer class

Parameters and Options

Select the instance in the World Outliner and navigate to its Details Panel. Here we will specify the properties of the **IMMSequencer** instance and fill in the information about the source video file, that was encoded with the Spin Digital HEVC codec.

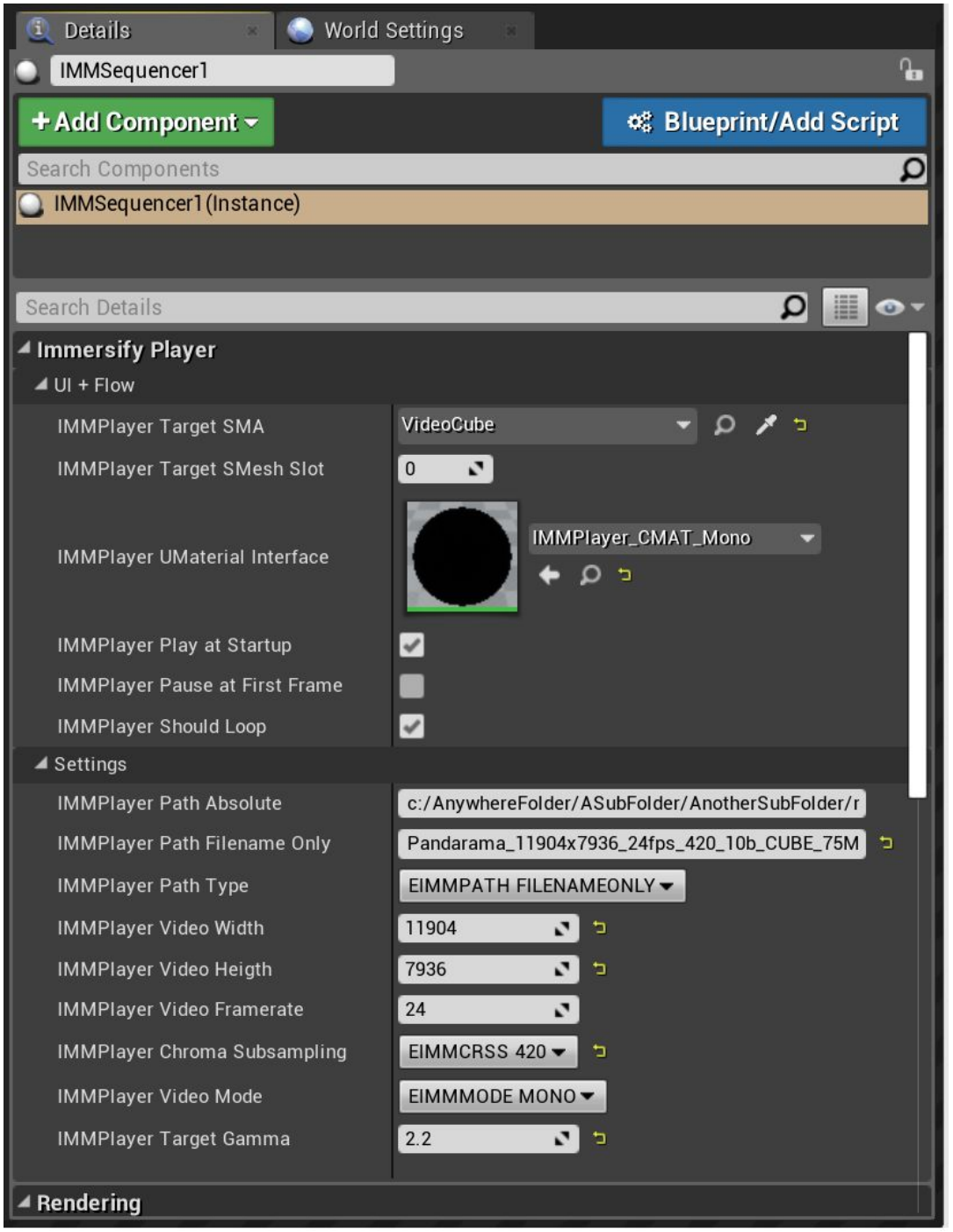


Figure 20: A Unreal Screenshot to see the available parameters of the IMMPlayer in Details Panel

Note: In the current version, the user is responsible for entering the correct properties for the video that shall be played (including width, height or chroma subsampling level of the video). There are no sanity checks on the parameters inserted into the Editor, and since the Immersify Video Player aims for maximum performance, it uses certain features of the Unreal Engine in a highly experimental fashion, hence incorrect input values will most often result in a misbehaving video or lead to an application crash.

The following properties can be defined and/or specified for/by the end user:

- **Settings/Path Type**: Selects the provided path to the video - either an absolute path or a filename only
- **Settings/Path Absolute:** An absolute system path e.g *C:/Directory/SubDirectory/myvideo.hevc*
- **Settings/Path Filename Only**: The name of the file of a video, e.g. *myvideo.hevc***,** which must be placed in the *<Unreal Project Directory>/Video/* folder.
- **Settings/Video Width**: The width resolution of the video in pixels.
- **Settings/Video Height**: The height resolution of the video in pixels.
- **Settings/Video Framerate**: The default playback frame rate of the video.
- **Settings/Chroma Subsampling**: The chroma subsampling (YUV) level of the video, options are 444, 422, 420.
- **Settings/Video Mode**: The Stereo Mode of the video, options are Monoscopic, Over-Under (Top-Bottom) Stereo and Side-By-Side Stereo. This setting is currently reserved for a future update, please select the Stereo Mode by using the appropriate Material in the "UI / Flow / IMMPlayer UMaterial Interface" field.
- **Settings/Target Gamma:** This setting is currently reserved for a future update regarding Unreal Engine's color management.
- **UI + Flow/Target SMA:** Assign the Static Mesh Actor on which the video should be rendered
- **UI + Flow/Target SMesh Slot:** Assign the target Material Slot of the Static Mesh Component of the Static Mesh Actor on which the video should be rendered
- **UI + Flow/UMaterial Interface:** Assign the Unreal Material Shader through which you would like to render the video. Prepared unlit-type shaders can be found in the Plugin's Content Directory.
  Please note that you can select the type of Stereo Mode for the video in this reference slot, by assigning the corresponding Material. Options are Monoscopic, Over-Under (Top-Bottom) Stereo and Side-By-Side (Left-Right). Additional materials can be created and added to the setup, with any desired properties, using the provided templates as the starting point.
- **UI + Flow/Play At Startup**: Should the video start playing immediately when the game level has finished loading and gameplay starts? (default is true) If false, the player has to be started explicitly by calling the Start() function from C++ or Blueprint.
- **UI + Flow/Should Loop**: Should the video start over again when it reaches its end? (default is true)

- **UI + Flow/Pause At First Frame**: Should the video auto-pause on its first frame? (default is false)

The core functionality is propagated to the Blueprint System, so that the video player can be used by a designer or coder who does not need to have C++ knowledge.

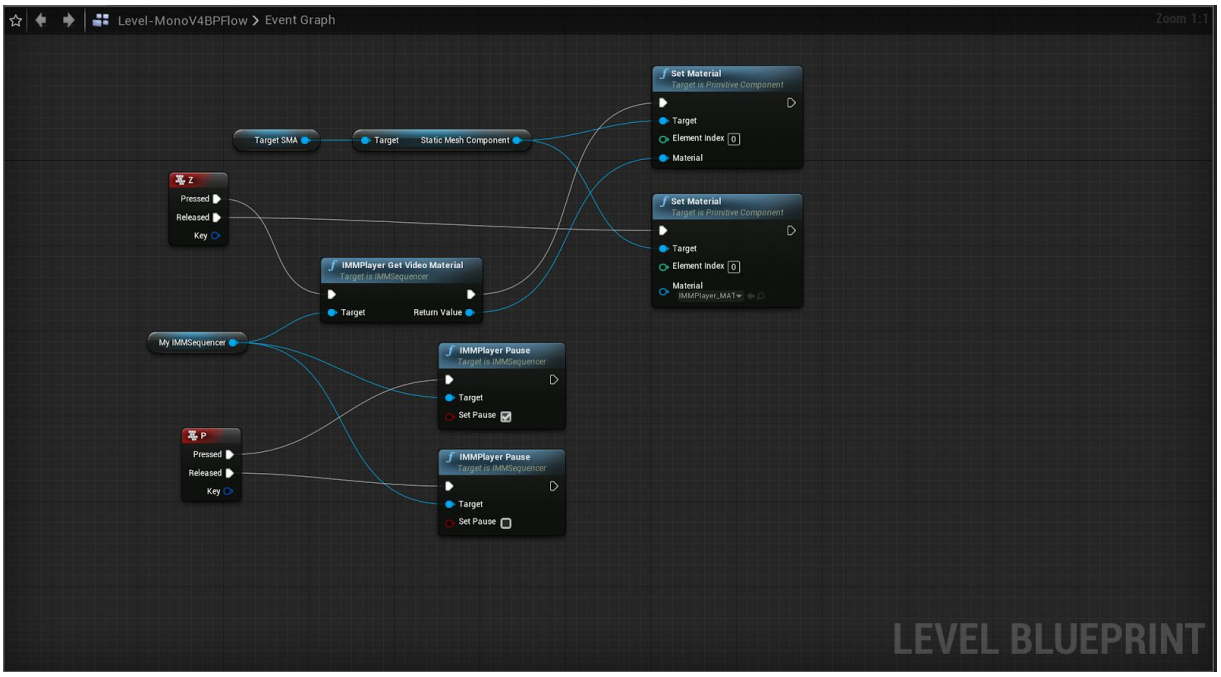


Figure 21: Screenshot of UE4 Blueprint that demonstrates using / controlling the Immersify Player (getting & assigning the video material and pause and unpause the video)

Figure 22: A Screenshot in UE4 of the IMMPlayer's Demo Scene, that is connected to the previously shown Blueprint

As the video-output is managed via UE4 Material, a designer is able to put this material to any mesh to display the video on any surface.

7.3 Results

The result of this project is a high end video plugin for Unreal Engine 4 that enables developers and end-users to play videos with very high resolutions (up to 16K x 16) in mono and stereo (with active stereo glasses or head mounted displays) and in the form of both planar and 360° videos.

7.4 Specification of media playback systems

The system specifications required for the Unreal Immersify Player are identical with the one from Section 5.4 (Specification of media playback systems), as this does the decoding in the background. Generally spoken, systems with more CPU cores are performing better. In our test with the Deep Space 8K (Windows 7 SP1 64 Bit, 2x Intel XEON CPU ES-2687W (2x8 cores), 4x NVIDIA Quadro P6000, 64 GB of RAM) we were able to playback 11K x 11K, 4:2:0 chroma subsampling, stereo videos with 30 FPS.

7.5 Advances over the State of the Art

There are a number of various video-handling solutions available for the Unreal Engine, either as a part of the standard Engine distribution or as paid professional plugin products available on the Epic Marketplace. We will show in the below comparison that the Unreal Immersify Player addresses a certain niche within this space, in such a way that no other offering comprehensively does.

We would like to emphasize again at this point that the Unreal Immersify Player offers, in a tightly integrated way, both a highly efficient video codec solution and a high-level framework for dealing with stereo and 360° video formats, while no other existing Unreal Engine product does this.

In comparison with the Unreal Windows Media Framework plugin[27] (short WMF, also referred as native video player), which is part of the standard engine distribution, or video plugins, that are available on the Unreal Engine/Epic Marketplace, the Immersify Video Player is capable of playing videos with higher resolutions and more resource-efficiently than the default players, and generally offers a better performance with regards to fast and stable frame rates. The WMF Unreal Framework also allows using the "HAP Codec"[28], which makes it possible to play high resolution videos with up to 60Hz, but has the tradeoff of using a lot of disc space and the requirement of a fast storage medium (like a SSD). However, at the moment, using the HAP Codec in WMF (and therefore UE4) is limited to 4K resolution.

The "Blackmagic & AJA "Codecs" (Unreal Engine 4 Professional Video I/O)"[29] are professional video editing solutions, which support 8K resolution video importing and exporting, by using special video capture hardware cards that are specified on the Unreal Engine documentation page. However, they are not suitable for realtime playback of video, their main purpose is to handle Unreal-generated footage within a VFX-compositing and video-editing work pipeline.

Plugins like the "UNAmedia Stereo Panoramic Player"[30] from the Epic Marketplace are intended for the playback of stereoscopic video and 360° images within Unreal. However, they are, without exception, based on the native WMF player, and therefore inherit all its limitations.

The Table 11 summarises the comparison of the described players. The note "on supported hardware" in the comparison table shall make it clear that a strong CPU is required to play high resolution videos with 30Hz or 60Hz smoothly. The note "on very special hardware only" means that this feature can only be used, if hardware specified in the documentation is used. The note "requires additional programming" means that this feature cannot be used out of the box, but it can be implemented on the base of this player.

[27] The Unreal WMF Player:
https://docs.unrealengine.com/en-US/Engine/MediaFramework/index.html
[28] HAP Codec:
https://docs.unrealengine.com/en-US/Engine/MediaFramework/TechReference/index.html
[29] Blackmagic & AJA Input: https://docs.unrealengine.com/en-US/Engine/ProVideoIO/index.html
[30] UNAmedia Stereo Panoramic Player:
https://www.unrealengine.com/marketplace/en-US/product/stereo-panoramic-player

Table 11: A comparison between the Standard Unreal Video Player, the UNAmedia Stereo Panoramic Player and the Immersify Player focussed on Immersify related content

Comparison	Standard Unreal Video Player	UNAmedia Stereo Panoramic Player	Immersify Player
1080p / Full HD	✓	✓	✓
4K / UHD	✓	✓	✓
8K	✓ (on very special hardware only)	✓ (on very special hardware only)	✓ (on supported hardware)
16K	✗	✗	✓ (on supported hardware)
VR / Stereo Support	✓ (requires additional programming)	✓	✓
360° Content	✓ (requires additional programming)	✓	✓

8 PSNC 8K 3D Media Player

8.1 Concept

One of the main purposes of the 8K 3D video wall installed at PSNC in New Media Laboratory is the presentation of high quality and high resolution video content, i.a. those developed in the Immersify project. It is used in various scenarios, in experiments with creating an immersive multimedia environment, evaluating 8K and 3D recordings and verifying innovative content creation techniques, as part of the studio scenery or simply for the entertainment of invited audiences.

During setting up the visualisation environment in 2015 it was decided to integrate display hardware (Barco e2, MCM50 and projectors) with powerful PC player based on SpinPlayer as a media source for the video wall processing system, however also other codecs such as TICO or JPEG2000 were evaluated and considered. The cooperation established between PSNC and Spin Digital, and the development in the Immersify project, resulted in that the SpinPlayer met all mentioned objectives and enabled the most flexible and rapid solution for experimentation for high resolution and immersive appliances. The capabilities offered by the player largely cover the demand, but several features were added to integrate with the specific infrastructure and improve convenience in some scenarios. These are:

- Auto configuration and switching between 2D and 3D. Various modes – with the best example of 2D vs active stereoscopic mode – require reconfiguration of both the hardware and the software settings. A “one button” solution to switch corresponding hardware/system setup with player settings was included that enabled switching the Barco e2 processing system and Crestron matrix to 2D or 3D mode.
- Web interface for remote control – the existing network interface in the player has been used and reworked to suit PSNC applications.
- Kinect integration - the ability to control playback using gestures has been added.
- Additional devices synchronization - integration of the new 8K system with previous projects developed in PSNC (e.g. using ImmersiaTV DVB-CSS protocol implementation for multi-stream synchronization), allowing synchronized 8K playback and additional information on complementary devices e.g. tablets or smartphones.

8.2 Implementation

Auto configuration and switching between 2D and 3D.

In its primary configuration, PC for PSNC 8K video wall provides output of the image for left and right eye separately on each of two P5000 Quadro graphic cards. The NVidia Mosaic technology is used, where 4 DisplayPort outputs of each GPU are combined into one 8K Windows desktop, resulting in 2 desktops of full 8K (8192 x 4320) resolution each (see also 8.4 Specification of media playback systems). In 2D mode only one desktop is projected onto the screen, and in 3D mode both desktops are projected alternately.

To switch between these modes SpinPlayer renderer configuration has to be altered respectively. In 3D mode the renderer splits the video into top and bottom half and renders these on both available desktops (*Tiled mode*). In 2D mode, on the other hand, the whole picture is rendered to the left desktop only (*Single mode*). Switching the setup of the processing and projection hardware is

integrated with the PSNC cinema room control system operated via Crestron Panel. To trigger corresponding configuration of PC and SpinPlayer change, a solution that captures the information from the Crestron system was implemented.

Auto configuration and 2D/3D switching of SpinPlayer is implemented as a node.js application. This allowed us to easily integrate all the required features in one application. The application communicates with video wall display and control hardware to obtain information about current display mode selected by the user. This information is later used to load pre-prepared configuration sets for SpinPlayer. The process consists of the following sequence of actions:

- User selects video wall operation mode (2D or 3D) on Crestron control panel,
- Crestron Controller reconfigures display hardware, i.a MCM processors, for selected mode,
- Application queries (periodically) one of MCM processors about its display mode (2D/3D),
- If display mode change was detected, the SpinPlayer instance is terminated,
- Appropriate SpinPlayer Settings are loaded into Windows Registry,
- If SpinPlayer was running before change, new instance is started (on startup it is loading its settings from Windows Registry),
- SpinPlayer is ready to play content in selected mode.

The process has been depicted on Figure 23.

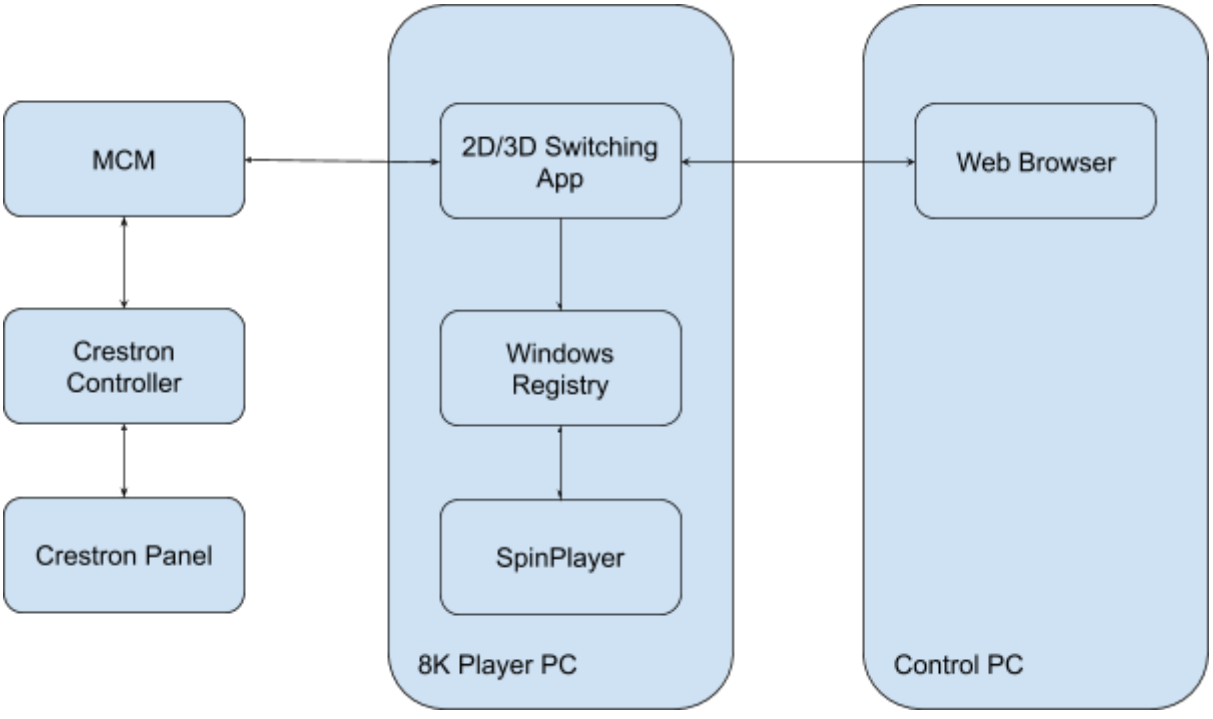


Figure 23: Schema of communication in playback modules configuration

In addition to its automatic mode, the Application serves a simple web interface that allows manual selection of SpinPlayer configuration, as well as control over NVidia's mosaic configuration of the player.

Figure 24: 2D/3D switching application web UI

Web interface.

Customized web interface (Figure 25) is one of the additionally implemented tools that improves and simplifies control of SpinPlayer. The PC operating the video wall is located in a different room than the wall itself. The web interface allows controlling media player and PC through the network via a responsive webpage. Using any device with a touchscreen in the same network (e.g. phone or tablet), a user can remotely control the mouse and the keyboard. Moreover, video files can be selected and opened quickly, as well as added to the playlist without moving the mouse cursor. Web interface allows reorganizing and saving the playlist too. User interface always shows basic media player controls (Figure 26) like play, stop, fullscreen, seeking and volume control.

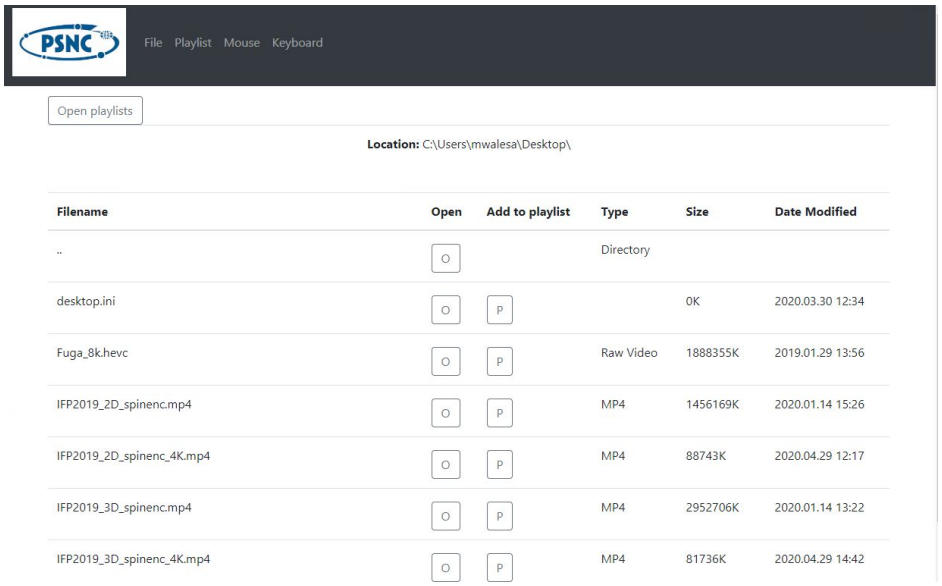


Figure 25: Example panel of web interface (playlist view)

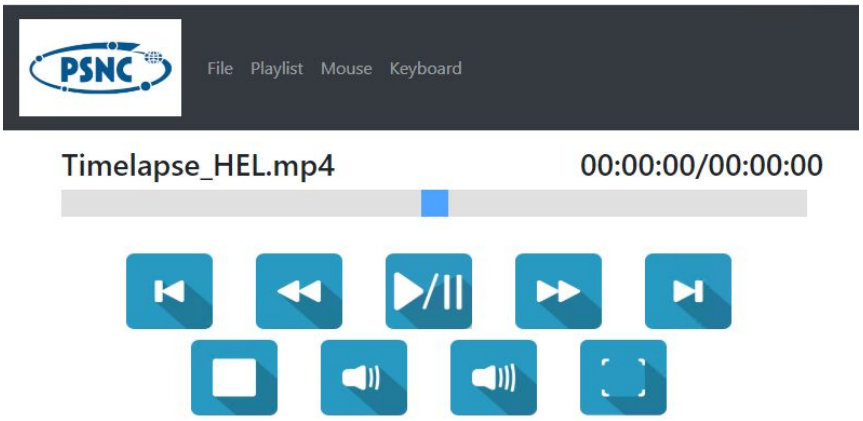


Figure 26: Control panel in web interface

Kinect integration.

In the course of the conducted experiments it turned out that a significant improvement would be to introduce control of the displayed content by means of gestures. Several methods were tested (e.g. Leap Motion), but the most promising solution was Kinect and the application for controlling SpinPlayer remotely by hand gestures was implemented. That is an attractive way of demonstrating new technologies. The gestures and their meaning can be defined depending on the planned scenario, for example a user can swipe his hand to switch video or raise and open both hands (as on the Figure 27) to start playing video.

This application is based on Microsoft KinectSDK and HTTP communication. Kinect technology allows to recognize and map the human's body position onto a virtual skeleton and further match it with the learned set of gestures. We have prepared a database of joints bends associated with gestures and player controls. Kinect application recognizes various gestures and sends corresponding messages to the player's web interface (e.g. volume up or play/stop).

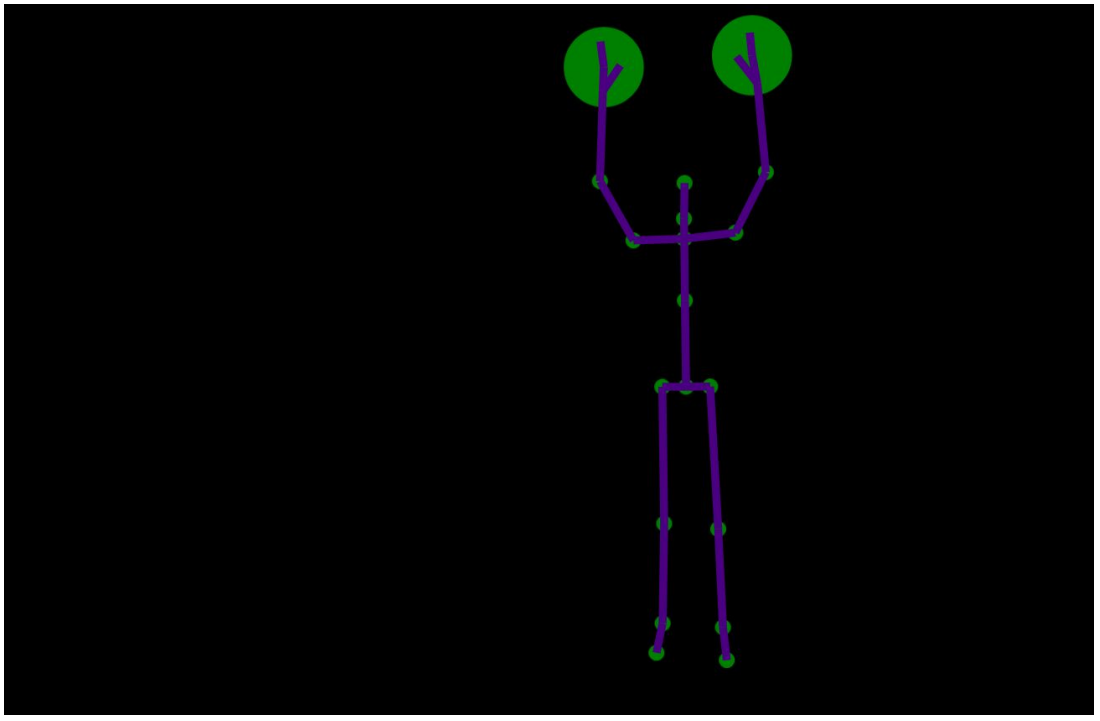


Figure 27: Recognized skeleton of play gesture

Additional devices synchronization.

In the PSNC New Media Laboratory, before the beginning of the project, operated the system developed in the ImmersiaTV project[31], that allows for synchronized playback of additional materials on other devices, e.g. tablets, smartphones, HMD, together with the main video on TV.

One of the objectives of the works in Immersify was to integrate a new high definition player with such a system. In this solution, the 8K screen becomes the main medium to which other devices adapt. The server supporting the screen makes available in the local network information about the currently played material together with the network reference clock. Any application that implements these protocols can use this information.

Preliminary work in this area and tests have already been signaled in D3.2 - Report on Decoder and Video Rendering for VR - Part 2. However, the goal was to fully integrate the solution in the same player used in the laboratory. In addition the same solution, with slightly increased requirements has also become the basis for PSNC Cave Media Player and implementation details are described in the next section.

A client application for secondary devices e.g to play 360 movies on a HMD was already available in PSNC. However an additional web application, easily customizable for given content, was developed to provide additional textual and graphical information synchronized with the progress of the story. It allows also controlling the playout, e.g. changing the chapters or making any other decisions defined for the content, which greatly supports course of interaction in non-linear storytelling scenarios. Figure 28 shows a sample view of a web application supporting the Poznań Cathedral movie.

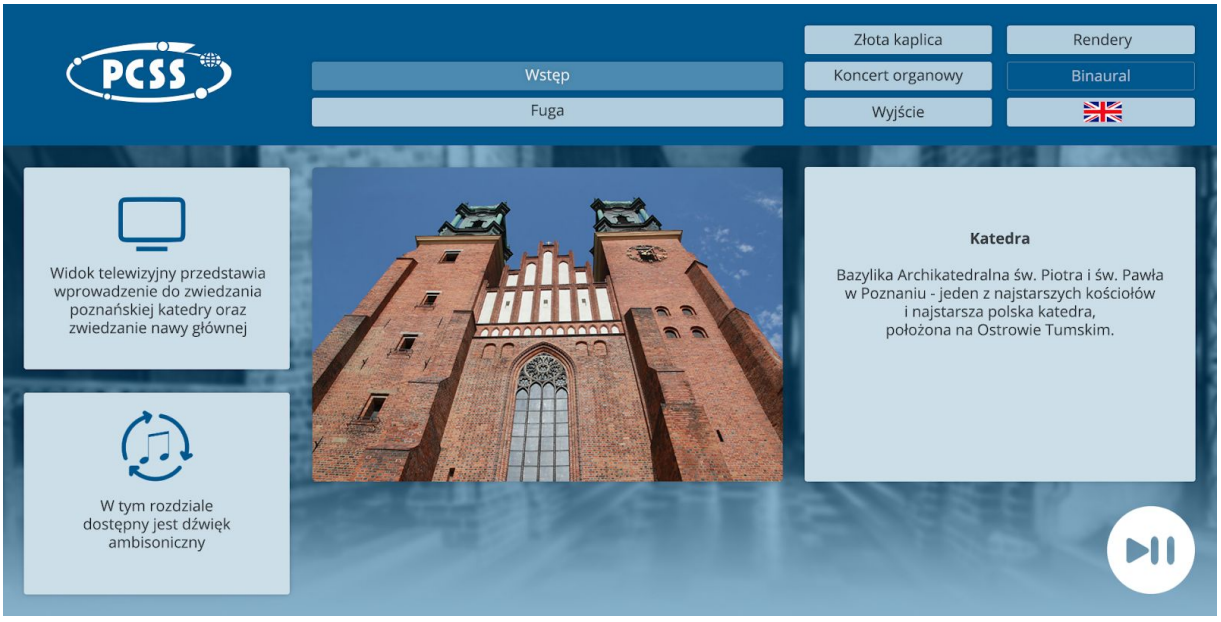


Figure 28: An example view of the web application available together with an 8K nonlinear movie about Poznan Cathedral

[31] H2020 ImmersiaTV project, grant agreement No 688619, http://www.immersiatv.eu

8.3 Results

As a result of the integration work, the visualization installation in the PSNC has been extended with unique features enabling the presentation of high quality film recordings. It also offers the possibility of interactive scripts, experiments with new formats and techniques. It can be conveniently configured and controlled to adapt to the envisaged presentation scenarios.

Figures 29 and 30 show internal PSNC demonstration of system capabilities: 8K screening with two associated mobile devices (for presentation purposes TV-sets were used): one with auxiliary textual and photo content, the second with 360 stream. Additionally several tablets and HMDs were used by the audience.

Figure 29: Internal demo for employees and invited guests. PSNC 8K 3D video wall working with the new player. Main screen with 8K content (center), auxiliary content (left-side) and 360 video (right-side) - all of them synchronized

Figure 30: Internal demo for employees and invited guests. PSNC 8K 3D video wall working with the new player and additional devices such as mobile phone inside google HMD

8.4 Specification of media playback systems

The 8K 3D 60p wall installed in New Media Laboratory is composed of 12 BARCO projectors in rear projection configuration and it is supported by BARCO processing system based on e2 and MCM-50 units. The general architecture is depicted on Figure 31 and the dimensions and resolutions have been presented in the table below.

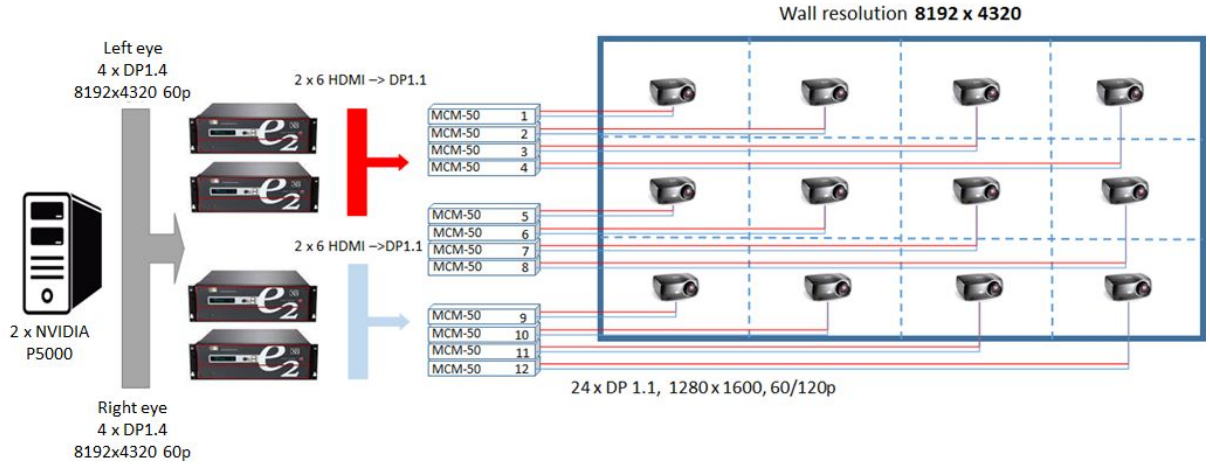


Figure 31: General architecture of the 8K 3D rear projection wall

Table 12: Specification of 8K 3D projection wall

Screen	Rear projection screen
Physical Dimensions	6m x 2.8m
Total resolution	8192 x 4320 per eye
Max. Framerate	60Hz (120Hz for 2D)
3D technology	Active
Display devices	12x BARCO F50 projector (4x3 layout)
Input processing	4x BARCO E2 video processing system 12x BARCO MCM-50 warping and blending system
Video source	**PC workstation with GPU**
CPU	2x Intel Xeon Platinum 8168
Memory	192GB
Output	2x GPU NVIDIA Quadro P5000
System	Windows 10 Pro 64bit
Software	SpinDigital Player

In this setup four Barco e2 processors are grouped by two, each pair is stacked and works as a single virtual device serving 8K image. Each pair of e2 devices processes one 8K image - one for left and one for right eye. It accepts video signals from DisplayPort, HDMI or SDI interfaces and generates proper image distribution for the projectors. In between e2 units and the projectors, the set of twelve BARCO MCM-50 boxes is installed which are responsible for blending, warping and color matching for F50 projectors. The MCM-50 serves as an input processor for the F50 WQXGA series projectors allowing sub-pixel geometry correction, color correction, brightness monitoring and electronic blending up to 120 Hz. This box enables several multi-channel configurations with multiple F50 projectors at native 120 Hz frame rates. The MCM-50 handles both mono and stereo sources up to 120 Hz. Native WQXGA Active or passive stereo sources are converted into an active flicker-free stereo signal up to 120 Hz. Four DisplayPort 1.1 inputs allow up to 2 sources to be connected in two columns of 1280 by 1600 pixels offering a full 2560*1600 pixel canvas. At the end of the video chain there are twelve Barco F50 lamp-based single-chip DLP projector that combines native WQXGA (2560 x 1600) resolution with active 3D stereo and high frame rates. The F50 boasts four different digital input sources and a wide range of expansion modules and high-quality projection lenses. In our configuration each projector is connected with a dedicated MCM-50 box using two DP1.1 interfaces and additional optical extenders.

8.5 Advances over the State of the Art

Immersive systems and environments are becoming very popular due to the increasing technological maturity that ensures adequate image and sound quality. One of the paths of development of this area is the creation of complex installations (CAVEs, Domes) allowing for the participation of many people in one immersive event. Such large scale installations, often called Location Based Entertainment spaces, are now rapidly developing, mostly showing VR experiences dedicated for common use by many visitors.

The industry is moving towards the creation in cinema theatres, museums, science and cultural centers. The technological solutions used to build such systems are very diverse. Although most of the large scale immersive installations share some common features (high resolution large screen displays, surround audio systems, etc.) but still they are very heterogeneous.

PSNC, by integrating SpinPlayer into its installation as the baseline media player for immersive applications, increases its independence from the infrastructure. With these efforts it introduces new standards to the industry allowing compatibility between different systems. With SpinPlayer the installation supports HEVC content encoding, with 8K resolution, 3D and high frame rates, as well as ambisonic sound.

PSNC establishes a reference laboratory of New UHD Media Services, where device vendors, software companies and research institutes, as well as artists and creators may test their solutions. The 8K 60p 3D 6-meter-wide display wall is a unique installation that enables scientists, researchers and SMEs conducting advanced research and developing applications for high level visualisation infrastructure and tests with the use of target systems.

9 PSNC Cave Media Player

9.1 Concept

In PSNC an immersive CAVE visualization space was created in the form of a cylindrical display. CAVE2 - as it's called in comparison to another traditional cubic CAVE1 in PSNC - consists of 45 curved adjacent Samsung TVs arranged in 3 rows. Potentially it allows to display images with a resolution up to 57600x6480, however such a resolution requires a powerful rendering cluster. It is a scalable display - the screens are divided into 5 segments supported by separate servers. It is also possible to work in other hardware configurations, covering only part of the space and using a smaller number of servers. For daily use, the final resolution was decreased, so it can be driven by just 5 PCs equipped with several graphic cards. The schematic view of CAVE2 has been depicted on Figure 32 and the real photo is shown on Figure 33.

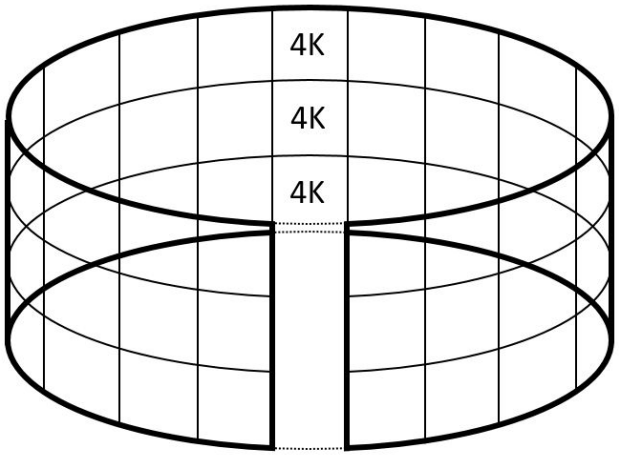


Figure 32: Schematic view of the CAVE2 installation composed of 45x 4K TV-sets

Figure 33: Part of CAVE2 installation

Works have been carried out to integrate the solutions used in the Immersify project with the above mentioned hardware architecture and allow for playback of very high resolution movie panoramas in immersive CAVE environment. Similarly to 8K 3D Media Player, the basis of the adopted solution is Spin Player. However, the key issue is to synchronize the playback in instances of applications running on servers supporting different segments of the display.

As noted in the previous section (and earlier in D3.2 - Report on Decoder and Video Rendering for VR - Part 2), PSNC has been working on a synchronization protocol that was extended to support scalable display. Implementation details and results are described in the following sections.

9.2 Implementation

Implementation of PSNC scalable display is based on a pre-existing solution developed in ImmersiaTV project[32], allowing synchronized multi device non-linear storytelling. Integration of Immersify high resolution player with that system enabled new capabilities of PSNC visualisation installations.
Before documenting the architecture and configuration of solutions developed in Immersify project it is necessary to briefly introduce elements and terms from the system we have been integrating to:

- **Session Manager** - network service that acts as the center element of synchronized playback setup. It Implements functionality of :
    - **DVB-CSS-CII** server - maintains playback sessions, provides information about content currently being played (*contentUrl*, *chapterId*) as well as urls of other required services (TS server, WC server),
    - **DVB-CSS-TS** server - provides timeline synchronization for current playback session,
    - **DVB-CSS-WC** server - provides wallclock synchronization information for clients.
- **Content XML file** - describes structure of playback: chapters, scenes, media files, transitions between scenes and chapters for all types of devices: TV, HMD, Tablet. This format allows to create complex contents combining multiple media files (mp4 or MPEG-DASH) for linear or nonlinear storytelling.
- **Content List** - json file containing list of entries, each describing one content with its name, url, thumbnail.
- **Content Server** - web server hosting content list, media files as well as content description XML files.
- **Session** - group of players in a common environment that share content selection and timeline synchronization and its representation on Session Manager instance.

Developed components
Implementation of multiplayer synchronization solution consists of three main parts:

- **SyncClient.dll** - Main element of solution, it communicates with Session Manager to obtain information about: content to play, timeline synchronization, and wallclock synchronization. Thanks to this information and its ability to fetch published content list and parse selected content XML file, it is able to determine which media file is required to play at a given point. It detects changes in timeline synchronization (DVB-CSS-TS protocol) to emit updated synchronization information to the player. As DVB-CSS-WC client periodically computes the difference between its own clock and one provided by Session Manager, this information in

[32] H2020 ImmersiaTV project, grant agreement No 688619, http://www.immersiatv.eu

conjunction with players feedback is used to fine tune clock values returned to the player for rendering timing and allows synchronized playback on multiple machines.

- **DVBSyncClock.dll** - serves as DirectShow wrapper, using clock data provided by SyncClient.dll, implements DirectShow Clock Filter for player's DS graph and provides DS Filter Property Page for easier control. It passes control information (file open, file seek, file stop, set volume) from SyncClient.dll to SpinPlayer and synchronization feedback information in the other direction.
- **SpinPlayer integration** - changes required to use DVBSyncClock.dll as a clock source for filter graph instead of default clock filter. Implementation of player specific functions like file open, close and seek.

New Operation Modes

With above modifications SpinPlayer can operate in additional two scenarios:

- **ImmersiaTV Content Player** - In this mode ImmersiaTV compatible content can be enhanced by 8K resolution TV playback. This scenario requires all elements of ImmersiaTV setup i.a Session Manager, Content Server and ImmersiaTV compatible content, companion devices like HMD or tablet are optional. Features like interactive chapters, looping are supported for nonlinear storytelling and immersive, multi-device experience.
- **Simple File Playback** - In this mode only Session Manager is required, any HEVC encoded MP4 file can be played synchronously on multiple devices. This mode suits well for PSNC CAVE installation as it doesn't require preparation of content XML file, but still allows for synchronized playback, seeking and content (mp4 file) selection, as well as centralized control over multiple players.

Filter Dialog

As mentioned before, **DVBSyncClock.dll** implements Direct Show Filter dialog that allows control over synchronization filter as well as editing its configuration parameters. In this section the layout of said dialog is described.

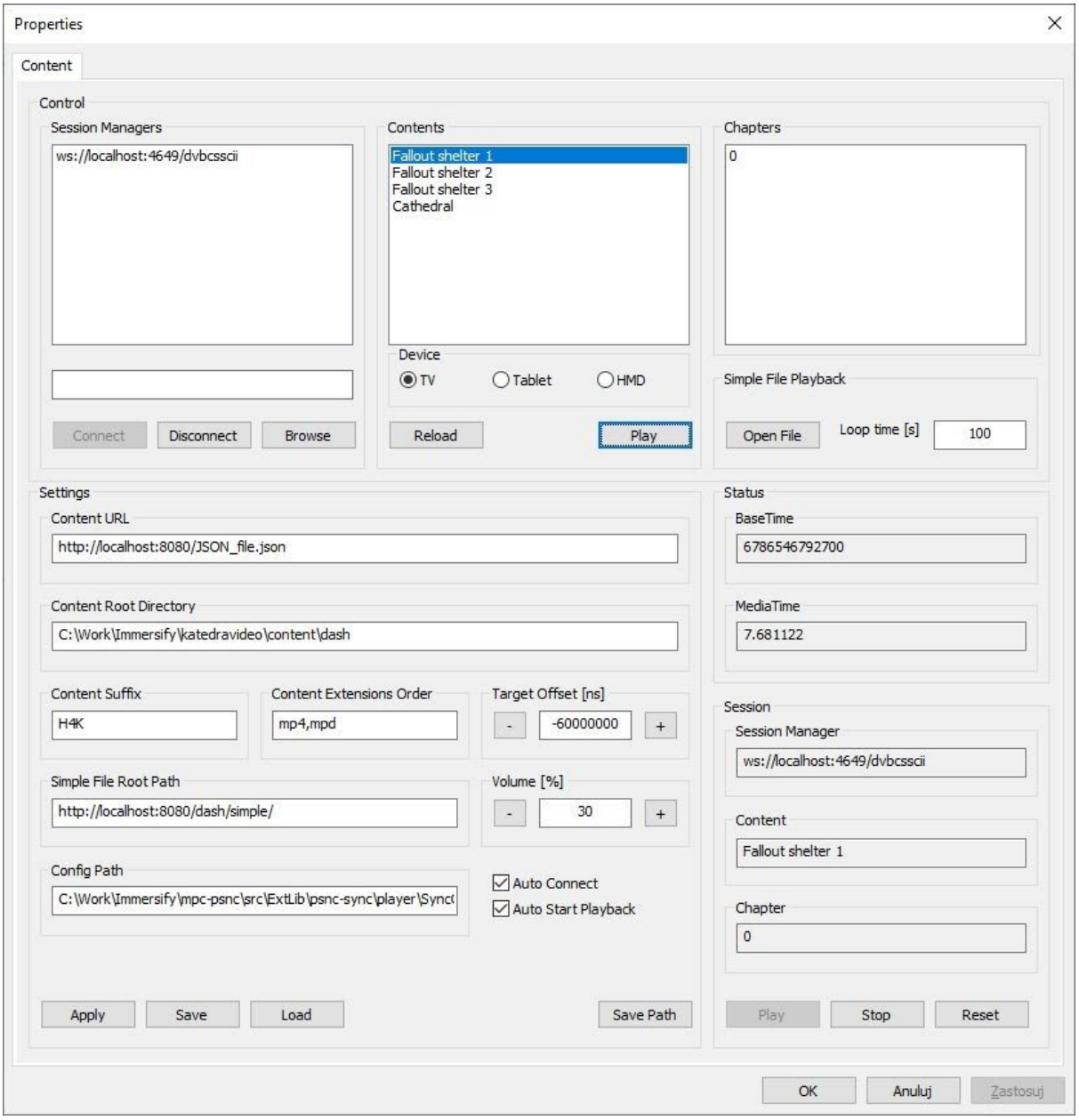


Figure 34: Synchronization properties filter dialog

- **Control**:
  - **Session Managers** - List of Session Managers discovered in local network
  - **Connect** - connect to selected Session Manager
  - **Disconnect** - disconnect from currently used Session Manager
  - **Browse** - Discover Session Managers in local network
  - **Contents** - list of contents available
  - **TV/Tablet/HMD** - selection content profile to play
  - **Reload** - reloads content from current Content URL
  - **Play** - Play selected Content/Chapter
  - **Chapters** - list of chapters in selected Content
- **Settings**
  - **Content URL** - url of json file containing list of available Content

- **Content Root Directory** - local directory where content media files are stored. For sake of efficiency it is advised to store media files locally on the player. This settings defines a path to the directory where SyncClient will expect to find them. If empty SyncClient will construct a full path to the media file based on content url.
- **Content Suffix** - suffix that will be added to mediaFile name. This allows switching between multiple variants of the same content without needing to edit the contents XML file.
- **Content Extension Order** - comma separated list of extensions to use for media files defined in content xml, SyncClient will try to use extensions to build a full Media File path in this order to check if a file is available (on disk or network).
- **Target Offset [ns]** - constant offset added to clock values provided for the player, this feature is useful for tuning synchronization with companion devices.
- **Simple File Root Path** - Url prefix that is used to construct and detect simple file mode. This path is also used by SyncClient to determine if the content that was selected by another player should be played in regular or simple file mode. The url is constructed as follows:
  URL = _SIMPLE_ROOT_PATH_ + "/" + _LOOP_SECONDS_ + "/" + _FILE_PATH_ + ".xml"
  e.g _URL_ = "http://192.168.1.10:8080/simple"+"/"+"200"+"/"+"D/Videos/Test.mp4"+".xml"
- **Config Path** - path (or url) of SyncClient config json file, this path is stored by player and used at startup to load SyncClient settings, this path is also used by Save/Load commands.
- **Volume [%]** - volume value used for file playback, volume set command is sent by SyncClient with each synchronization change (seek, file open).
- **Auto Connect** - if set SyncClient will try on startup to connect with Session Manager defined in config file.
- **Auto Start Playback** - if set SyncClient will start playback of content selected by any other device on the same session, if not set, content selection on another player will set selected contentId in **Content** filed, but the playback will not start.
- **Apply** - applies settings from property page to SyncClient instance.
- **Save** - saves current SyncClient's config to file.
- **Load** - loads config from file into SyncClient instance
- **Save Path** - requests the player to store Config Path. SyncClient config path is stored by the player, this command causes Spin Player to store it.

- **Status**
  - **BaseTime** - if playback is active shows current media file wallclock start position; changes on each seek.
  - **MediaTime** - if playback is active - shows position in seconds in the current media file.
- **Session**
  - **Session Manager -** url of current Session Manager
  - **Content** - ContentId of current session
  - **Chapter** - ChapterId of current session
  - **Play** - starts playback of active content
  - **Stop** - stops playback of active content
  - **Reset** - resets active content, all devices on the same session will stop playback and return to its content selection states.
- **Single File Playback**
  - **Open File** - opens File selection dialog and starts file playback in Simple File Mode
  - **Loop time [s]** - time in seconds after which file playback in Simple File Mode will be restarted.

9.3 Results

SpinPlayer was enhanced with additional modes of operation: **Simple File Mode** - multi device synchronization for PSNC CAVE2 and **ImmersiaTV Content Player** for nonlinear, immersive storytelling for contents like Poznań Cathedral presented in PSNC 8K Cinema.

Figure 35: Test of the synchronization in CAVE - 2 segments displaying the same mirrored content

9.4 Specification of media playback systems

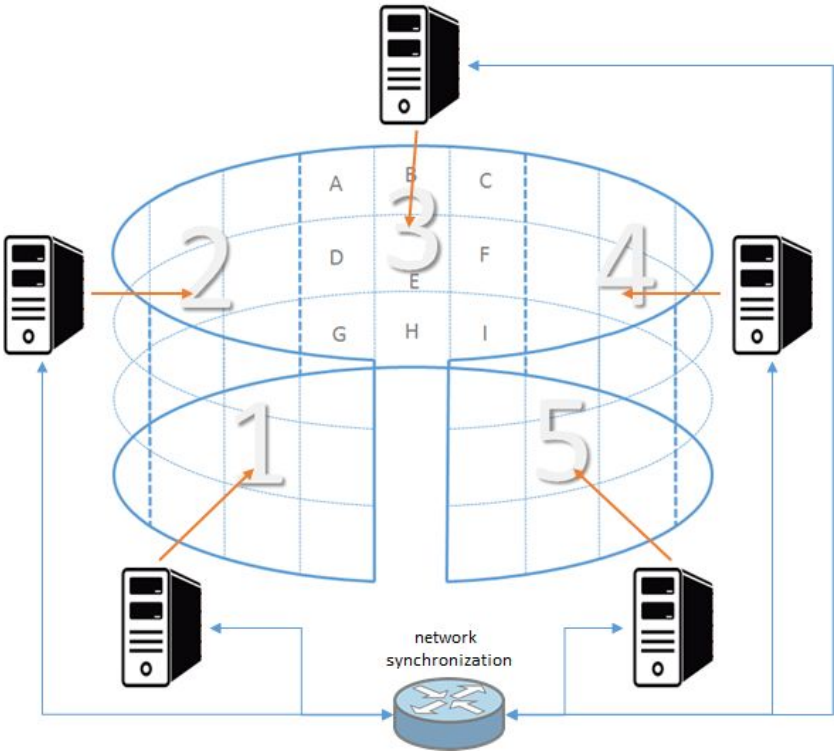


Figure 36: PSNC CAVE2 general architecture

Table 13: Specification of CAVE2 playback system

Screen		**CAVE2**
Physical Dimensions		diameter 7.74m, height 2.72m
Max. resolution		57600 x 6480
Supported resolution		28800x3240
Max framerate		60Hz
Display devices		45x curved 65’ TV - Samsung UE65JU7500L
Input layout		5 segments (3x3) with Nvidia Quadro sync
Video source		**5x PC workstation with GPU**
5x	CPU	Intel Xeon 2697-V3 2x14 cores
	Output	3x GPU NVIDIA Quadro K6000
	System	Windows 10 Pro 64bit
	Software	SpinDigital Player

9.5 Advances over the State of the Art

The idea of scalable displays allows building complex, but cost-effective installations. They can be easily adjusted to various requirements e.g. computational power. With the approach implemented in CAVE2, many interesting immersive space configurations can be designed. Integration with Spin Player provides the support of high quality content, with all the novel features important for the media player and implemented in the Immersify project to be available for the CAVE installation.

Proposed synchronization mechanisms guarantee flexibility and easily extensible setup. Not only does it provide the synchronization of screens forming a single surface, but the system remains open for integration with additional devices and applications, and development of innovative (e.g. interactive) scenarios.

Such capabilities make it a very unique installation, however our experiments show, that it can be successfully utilized in virtual reality and immersive spaces. Further development and popularization of this approach could ease the production of new content and applications and enable exchange of them and closer cooperation between visualization centers.

10 NVAB Dome Player

10.1 Concept

The *NVAB dome player* is an OpenGL-based video player with cluster/multi-machine and multi-viewport support, written in C++/GLSL. For video decoding, it utilizes the Spin Digital SDK. The *NVAB Dome Player* is actually not only supporting a dome theatre, though its primary use case is such a facility. It was generally designed to support any immersive multi-display environment which requires a multi-machine/cluster system to achieve playback of a resolution that matches the capabilities of the environment. That generally implies that features such as hardware synchronization, including frame locking, and network synchronization of states and user interaction need to be supported. Also, it is beneficial that the software itself can be configured for any immersive environment without the need to re-compile the application and make multiple custom versions. A single machine setup is still of interest for this software, thus multi-viewports and/or multi-window setups need to be supported as well.

10.2 Implementation

As NVAB already has experience working with local universities and other organizations across the globe with access to dome theatres, the case for using already established toolkits as a backend was strong. NVAB thus hired a developer with deep knowledge in building software for immersive environments. The current state-of-the-art at the dome in Norrköping was utilizing an in-house toolkit, released as open-source, named SGCT (Simple Graphics Cluster Toolkit) for the bases of building and OpenGL based application from the ground-up. This toolkit has been used in large softwares (such as OpenSpace https://www.openspaceproject.com/) and project work for students and creators since many years back. The version of the SGCT[33] utilized was v2.9, the latest stable release[34].

The most important component in SGCT is the engine. The engine handles all the initiation, rendering, network communication and configuration handling. The user can bind functions (callbacks) to the engine to customize specific tasks. Callbacks for keyboard and mouse input are handled by the window handler GLFW. Bonded functions will be called in different stages in the rendering process illustrated in Figure 37.

Figure 37 details what happens during launch and runtime with an SGCT application. Every single new iteration starts with synchronization of parameters, from the master/server to all clients, which in the player are such variables as play/pause/playtime etc. After the synchronization, but before the draw calls (PostSyncPreDraw), the video is decoded (in separate threads) by calls to the SpinDigital SDK, and the decoded frame is uploaded to the GPU with multiple PBO:s (Pixel Buffer Objects), all asynchronous. Using multiple PBO:s ensure that the playback, decoding and uploading thus not stall, and these operations can still go on while the current rendering/frame is being performed. The upload is done in a hidden GLFW window which has a shared context with the main GLFW window where the visible rendering occurs, as well as the mapping of the decoded frame onto a specific virtual object.

[33] https://sgct.github.io/ - Up-to-date documentation.
[34] https://github.com/sgct/sgct/tree/release/v2.9 - Current used branch.

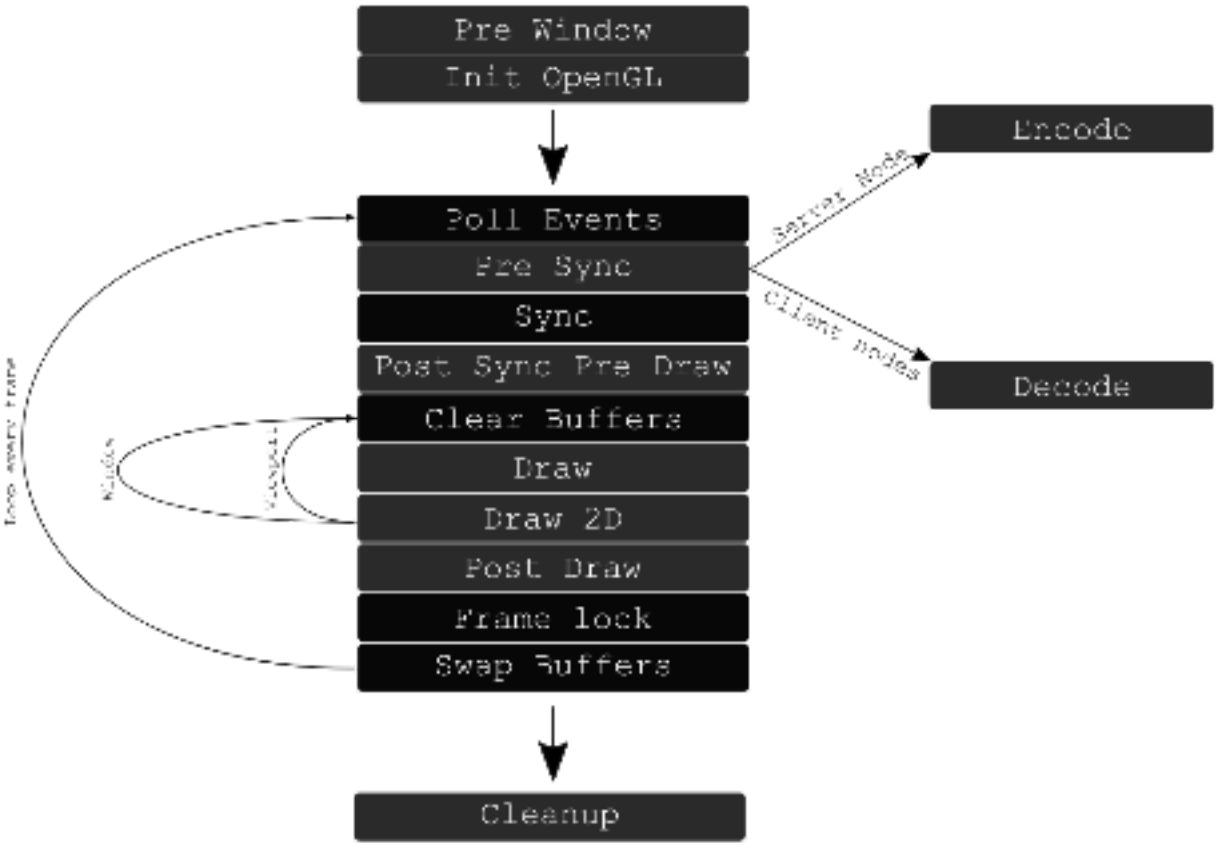


Figure 37: SGCT engine pipeline

The number of draw calls thus vary depending on the amount of viewports (even sub-viewports) that have been configured. The current dome setup is either one computer per projector, though 2 viewports per system, as the dome has 3D/stereoscopic capabilities.

There is also single system support (in mono), as one computer can quite easily be set up to run 6 displays in sync, though the total screen resolution, when using NVIDIA Mosaic or AMD EyeFinity was 12288x4320 (3x2 viewport setup).

While the number of viewports defines the number of draw calls, the decoded frame or frames need to be mapped accordingly. Depending on the video layout, the application renders and texturizes three different objects, a dome/hemisphere, a full sphere and a box/cube. All are textured on the inside, and a frame in fisheye format is mapped on a dome, an equirectangular frame on a sphere and a cubemap onto 6-sides of a cube/box. Typically about 256 segments is enough for a virtual dome and a virtual sphere to texture it properly. After all draw calls (in 3D) additional calls for overlays/2D/UI is made (primarily only used on the server/master) as well as a post draw call is made (not utilized in this application).

The video decoding is utilizing basically the same “Native Core Plugin” as described in Section 5.2.1, where Native Core classes utilize the Spin-Decoder SDK, where-as the Spin-Decoder runs in the background and decodes video data that has been encoded with Spin-Encoder.

10.3 Results

The NVAB dome player has been targeted for the dome in Norrköping, which is highly state-of-the-art at this time, with six powerful 6P-laser projector of DCI-4K resolution(4096x2160 @ 60Hz) per eye.

Multiple videos have been tested, such as sequences in 8Kx8K fisheye, freely available by ESO (European Southern Observatory, for instance https://www.eso.org/public/videos/illustris/). Figure 38 show-case a mono setup where 6 different viewports are configured according to the projector placements and field-of-views in Norrköping.

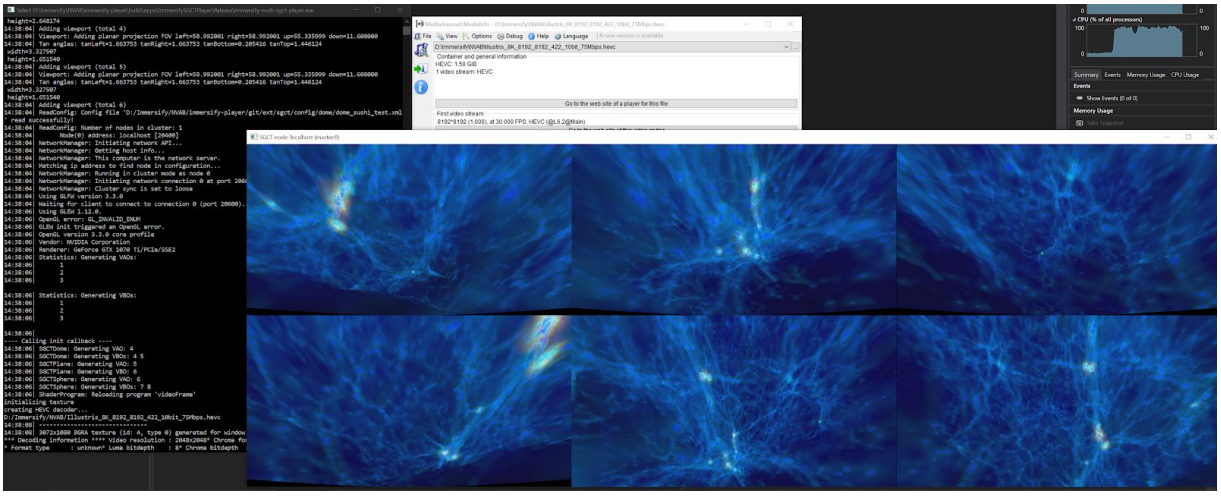


Figure 38: 8K x 8K fisheye movie decoded and mapped to 6 projection planes on a virtual dome surface, corresponding to the projection of the DCI-4K 6P Laser projectors. Video was encoded with specifications: HEVC @L6.2@Main, 8 192 x 8 192 pixels @ 30 FPS, YUV, 4:2:2, 8-bit, BT.709

Other fisheye videos from ESO along with content created from Immersify (such as OpenSpace sequences and the 360° *Poznan Cathedral*, produced by PSNC), were viewed in the dome.

Any 360° video, such as the *Poznan Cathedral*, can be interactively rotated during playback, such that the audience can view roughly 165° (the dome FOV) at the same time.

10.4 Specification of media playback systems

The single mono machine setup in Norrköping used with this player had 2 x Intel i9-9980XE (18-Core 3.0GHz) CPU:s as well as 2 x NVIDIA Quadro RTX 4000 cards, for 6 DCI-4K signal outputs, as well as 128 GB of RAM. The 6 machine cluster has 2 x Xeon E5-2637v4 (4-Core, 3.50 GHz), one NVIDIA Quadro P5000 GPU and 64 GB of RAM.

10.5 Advances over the State of the Art

Both single machine and multiple machine setups are very interesting. The main playback system currently used in day-to-day operation consists of 1 PC per projector, thus handling 2 DCI 4K signals each (in 3D-mode). As well, it supports only pre-splitted fisheye/fulldome videos.

Thus, it was of primary interest to see if playback of high-resolution videos from single systems were possible, thus making pre-splitting not necessary, and possibilities such as 360° videos more appealing, to interactively show-case these types of videos, which are becoming more and more common.

We were excited to see that the playback of 8Kx8K fisheye/fulldome movies, very high resolution, was possible from such a single system as described in 10.4, utilizing the dome player with the Spin Digital SDK. That has not been possible before, other than with HAP coded videos, which require more less decoding performance of HEVC, but are really huge in file sizes in comparison.

Also, with the dome player, 360 interactive rotation of video to 6 x DCI 4K views was possible, even with an 8K 360° video.

11 Task Status

11.1 Expected Results

The expected results for Task 3.5 are:

Complete media player with support for real-time playback of high quality panoramic formats targeting 8Kp120 and 16Kp60 resolutions and frame rates. Reference design for video playback for at least three different display architectures (PC with HMD, multi-display, and multi-projector). The outcome of this task will also be reported in D3.2. Roles: This task is led by AE. SD will integrate, test, and validate the optimized decoder and renderer into the media player for VR HMD, AE into the Deep Space 8K player, and PSNC into its multi-screening system. AE will also contribute to the integration of the HEVC decoder into the Unity3D engine and Unreal Engine 4. PSNC will integrate and test the media player in its multi-display and multi-projector visualization systems. NVAB will explore the adaption of the results to dome theatres.

11.2 Obtained Results

As was stated before, the main objective of this task is to integrate the enhanced HEVC decoder, video renderer, and 3D audio codec into a full-featured media player for immersive spaces. With the set of immersive players presented in this deliverable, the main objective is fully achieved.

Task 3.5 includes three main sub-tasks which are assessed individually:

1) *integration of the optimized decoder and renderer from T3.1, T3.2 and T3.3 into the target playback platforms, such as: media player based on "Media Player Classic BE" (SD), Deep Space 8K player (AE), the media server and 8K 3D network player (PSNC), Dome theater player of NVAB, and an Unity3D plugin wrapper.*

The goal of this subtask was fully achieved. The features developed in Tasks 3.1, 3.2, and 3.3 were integrated into an SDK called Spin SDK, which was used as the core for the immersive media players developed for different target immersive environments. The list of players and the corresponding section:

- Spin Digital Spin SDK, Spin Player, and streamplay - Section 4
- Ars Electronica Immersify Player for Deep Space 8K - Section 5
- Ars Electronica Unity3D Immersify Player - Section 6
- Ars Electronica Unreal Engine 4 Immersify Player - Section 7
- PSNC 8K 3D Media Player - Section 8
- PSNC Cave Media Player - Section 9
- NVAB Dome Player - Section 10

2) Porting the media player to multiple HMD displays including PC based (e.g. Oculus Rift, HTC Vive with Windows OS) and mobile-based ones (e.g. Samsung Gear VR with Android OS).

As explained in the section about deviations from the envisioned work, we gave more priority to the development of media players for large screen immersive environments such as Deep Space 8K, 8K video wall, cylindrical cave, and dome, rather than HMD and mobile devices. Nevertheless, we implemented limited support for HMD playback using the Unity 3D player integration framework, as

described in Section 6 “Unity Immersify Player”. This development allows the playback of high-resolution 360° video, using the HEVC decoder integrated into the Unity Immersify player on Oculus devices.

3) Specification of reference systems that include information about the configuration of the hardware, system software, and interconnection with TVs and projectors.

For each of the media players presented in this report a short description of the specifications of media playback systems has been presented. In addition, in Section 4 of the Deliverable 5.3 - *Report on QA and Content Preparation Guidelines*, more detailed examples of hardware settings for different use-cases envisioned in the Immersify project were presented.

Declaration

The information, documentation and figures available in this deliverable, is written by the members of *Immersify (Audiovisual Technologies for Next Generation Immersive Media)* project consortium and does not necessarily reflect the views of the European Commission. The European Commission is not liable for any use that may be made of the information contained herein.

This document contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.