Unity Immersify Player

In the following you will get an overview of all components of the Unity Immersify Player

Core Components

The core components / classes of 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 former is used for planar, the latter 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.

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.

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[1]
- **ImmersifyEquiRectangularVideo**: This is a 360° video player scene for equi-rectangular videos[2]

[1] An introduction to Equi-Angular Cubemaps: https://blog.google/products/google-ar-vr/bringing-pixels-front-and-center-vr-video/

[2] An introduction to Equi-Rectangular Cubemaps: https://medium.com/@onix_systems/how-to-use-360-equirectangular-panoramas-for-greater-realism-in-games-55fadb0547da

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

Implementation

The interface to the Immersify native plugin (dll), 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[3]).

- 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.

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

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

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

- 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);

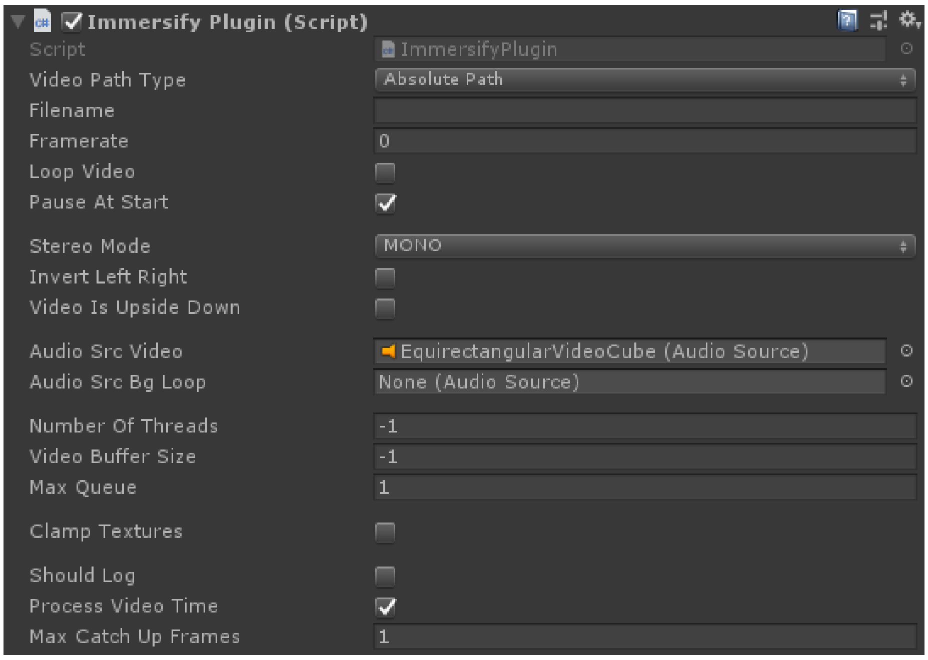
  This function registers a callback method that enables the C++ plugin to log via the Unity Editor interface. We made the experience that Unity runs unstable when using this feature. It can be used for debugging though.

Using the Unity Immersify Video Plugin

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

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.



Immersify Plugin (Script)	
Script	ImmersifyPlugin
Video Path Type	Absolute Path
Filename	
Framerate	0
Loop Video	☐
Pause At Start	✓
Stereo Mode	MONO
Invert Left Right	☐
Video Is Upside Down	☐
Audio Src Video	EquirectangularVideoCube (Audio Source)
Audio Src Bg Loop	None (Audio Source)
Number Of Threads	-1
Video Buffer Size	-1
Max Queue	1
Clamp Textures	☐
Should Log	☐
Process Video Time	✓
Max Catch Up Frames	1

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. Default is -1 and lets the plugin (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.

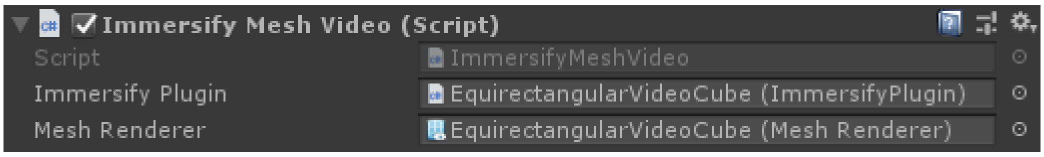
Clamp Textures: Required for some meshes, to set the render textures to *clamp* mode instead of *repeat* mode, to prevent visual artefacts.

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.

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.


Immersify Mesh Video (Script)

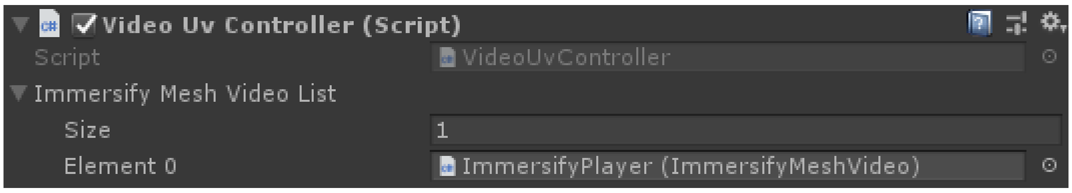
Script	ImmersifyMeshVideo
Immersify Plugin	EquirectangularVideoCube (ImmersifyPlugin)
Mesh Renderer	EquirectangularVideoCube (Mesh Renderer)

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.

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.


Video Uv Controller (Script)

Script	VideoUvController
Immersify Mesh Video List	
Size	1
Element 0	ImmersifyPlayer (ImmersifyMeshVideo)

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.

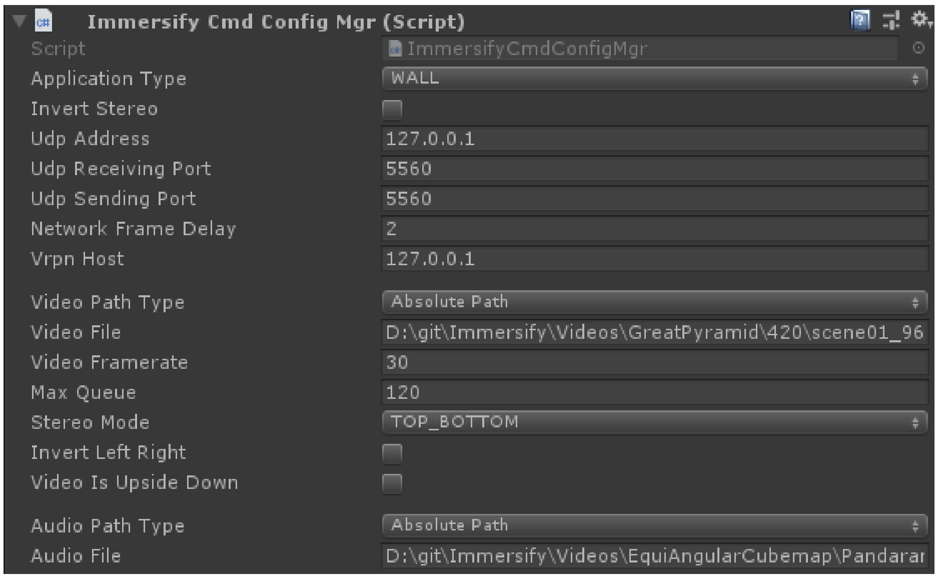
The UvController needs to know about the ImmersifyMeshVideo component. (It gathers several information from this binding class.) During rendering, the UvController changes the UV transformation, so that mono and stereo videos are displayed correctly. It thereby respects the stereo related settings that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

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[4].

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.

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



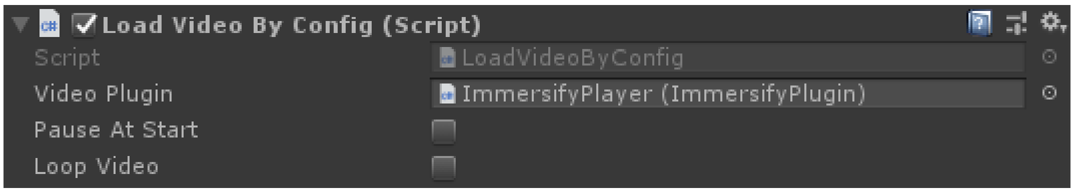
Script	ImmersifyCmdConfigMgr
Application Type	WALL
Invert Stereo	☐
Udp Address	127.0.0.1
Udp Receiving Port	5560
Udp Sending Port	5560
Network Frame Delay	2
Vrpn Host	127.0.0.1
Video Path Type	Absolute Path
Video File	D:\git\Immersify\Videos\GreatPyramid\420\scene01_96
Video Framerate	30
Max Queue	120
Stereo Mode	TOP_BOTTOM
Invert Left Right	☐
Video Is Upside Down	☐
Audio Path Type	Absolute Path
Audio File	D:\git\Immersify\Videos\EquiAngularCubemap\Pandarar

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.

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.



Script	LoadVideoByConfig
Video Plugin	ImmersifyPlayer (ImmersifyPlugin)
Pause At Start	☐
Loop Video	☐

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.