RENDERHEADS

AVPro Video

for Android, iOS, tvOS, macOS, WebGL
Windows Desktop, Windows Phone and UWP

Unity plugin for fast and flexible video playback

Version 1.5.6

Released 8 November 2016

Contents

1. Introduction
    1. Features
    2. Trial Version
    3. Media Credits
2. System Requirements
    1. Platforms not supported
3. Installation
    1. Trial Version & Watermark Notes
    2. Installing Multiple Platform Packages
4. Usage Notes
    1. Platform Notes
    2. Video File Locations
    3. Streaming Notes
    4. Augmented / Virtual Reality Notes
    5. Hap Codec Notes
    6. Transparency Notes
    7. Hardware Decoding
5. Quick Start Examples
    1. Quick Start Fastest Start for Unity Experts
    2. Quick Start Fullscreen Video Player using Prefabs
    3. Quick Start 3D Mesh Video Player Example using Components
6. Usage
    1. Getting Started
    2. Unsupported Platform Fallback
    3. Components
    4. Scripting
7. Asset Files
    1. Demos
    2. Prefabs
    3. Scripts
8. Scripting Reference
9. Supported Media Formats
    1. Android
    2. iOS, tvOS and OS X
    3. Windows
    4. Windows Phone / UWP
    5. WebGL
10. Support
11. About RenderHeads Ltd

Appendix A - FAQ
Appendix B - Version History
Appendix C - Roadmap

1. Introduction

AVPro Video is the newest video playback plugin from RenderHeads. We previously developed the AVPro QuickTime and AVPro Windows Media plugins for Unity. In this next generation of plugins we aim to create an easy to use, cross-platform video playback system that uses the native features of each platform.

1.1 Features

- Versions for iOS, tvOS, OS X, Android, WebGL, Windows, Windows Phone and UWP
- One API for video playback on all supported platforms
- 8K video support (on supported hardware)
- VR support (mono, stereo, equirectangular and cubemap)
- Transparency support (native and packed)
- Fast flexible video playback
- Unity Pro 4.6.9 and 5.x supported
- Free watermarked trial version available (download here)
- Fast native Direct3D, OpenGL and Metal texture updates
- Linear and Gamma colour spaces supported
- Graceful fallback in editor
- Components for IMGUI, uGUI and NGUI
- Easy to use, drag and drop components
- Streaming video from URL (when supported by platform)

1.2 Trial Version

We offer an unlimited trial version of AVPro Video for download from our website at http://renderheads.com/product/avpro-video/. The trial version has no missing features or time restrictions but it does apply a watermark to the rendered output. The watermarking does have a small performance impact which is only really noticeable on very high resolution videos. In Windows if the GPU decoding path is used when the watermark isn't displayed - instead every few seconds the video size will scale down.

1.3 Media Credits

BigBuckBunny_360p30.mp4 - (c) copyright 2008, Blender Foundation / www.bigbuckbunny.org

BigBuckBunny_720p30.mp4 - (c) copyright 2008, Blender Foundation / www.bigbuckbunny.org

SampleSphere.mp4 - (c) copyright Daniel Arnett, https://vimeo.com/97887646

2. System Requirements

- Unity 5.x or Unity Pro 4.6 and above
- iOS 8.1 and above
- tvOS 9.0 (Apple TV 4th Gen) and above
- OS X 10.7 and above, 64-bit only
- Android 4.1 (Jelly Bean, API level 16) and above (ARM7 and x86)
- Windows XP and above (32-bit and 64-bit)
- Windows 8.0 and above (32-bit and 64-bit)
- Windows Phone UWP 8.1 (32-bit and ARM)
- Windows Desktop UWP 8.1 (32-bit, 64-bit and ARM)
- Universal Windows Platform 10 (32-bit, 64-bit and ARM)
- WebGL compatible browser

2.1 Platforms not Supported

- WebPlayer
- Linux desktop
- Tizen
- Samsung TV
- Game Consoles (XBox*, PS4 etc)

** XBox One may be supported using UWP build option. We have not tested this though.*

3. Installation

1. Open up a fresh Unity session (to clear any locked plugin files)
2. Import the **unitypackage** file into your Unity project. If prompted to upgrade some scripts click Yes.

3.1 Trial Version & Watermark Notes

3.1.1 Watermark

If you are using a trial version of the plugin then you will see a watermark displayed over the video. The watermark is in the form of a “RenderHeads” logo that animates around the screen, or a thick horizontal bar that move around the screen. In Windows if the GPU decoding path is used when the watermark isn’t displayed - instead every few seconds the video size will scale down.

The full version of AVPro Video has no watermarks for any platforms. If you use one of the platform specific packages (eg AVPro Video for iOS, or AVPro Video for Windows) then you will not see the watermark on the platform you purchased for, but you will see the watermark on the other platforms. For example if you purchased AVPro Video for iOS then you will still see the watermark in the Unity editor as this is running on Windows/OS X, but the videos played back when you deploy to your iOS device will be watermark-free.

3.1.2 Updating from Trial Version

If you are upgrading from the trial version, make sure you delete the old /Assets/Plugins folder as this contains the trial plugin and could conflict. You may need to close Unity first, delete the files manually and then restart Unity and re-import the package (because Unity locks native plugin files once they are loaded).

You can check which version you have installed by adding an MediaPlayer component to your scene and clicking on the ‘about’ button in the Inspector for that component. The version number is displayed in this box.

3.2 Installing Multiple Platform Packages

If you are not using the full all-in-one AVPro Video package and instead have opted to purchase multiple individual platform packages then the installation must be done carefully, especially when upgrading to a new version.

If you have installed the iOS package then it will also contain plugins for all of the other platforms but with the watermark enabled. This means that if you then try to install another AVPro Video package it may not override the plugins correctly. Here is how to resolve this using the iOS and Android package as examples:

1. Open a fresh Unity instance (this is important as otherwise Unity may have locked the plugin files which prevents them from being upgraded)
2. Import the iOS package
3. Import the Android package, but make sure that you have the iOS native plugin file unticked (so that it is not overwritten)

   A similar process can be applied for other package combinations.

List of native plugin files:

- Android
  - Plugins/Android/AVProVideo.jar
  - Plugins/Android/libs/armeabi-v7a/libAVProLocal.so
  - Plugins/Android/libs/x86/libAVProLocal.so
- macOS
  - Plugins/AVProVideo.bundle
- iOS
  - Plugins/iOS/libAVProVideoiOS.a
- tvOS
  - Plugins/tvOS/libAVProVideotvOS.a
- WebGL
  - Plugins/WebGL/AVProVideo.jslib
- Windows
  - Plugins/WSA/PhoneSDK81/ARM/AVProVideo.dll
  - Plugins/WSA/PhoneSDK81/x86/AVProVideo.dll
  - Plugins/WSA/SDK81/ARM/AVProVideo.dll
  - Plugins/WSA/SDK81/x86/AVProVideo.dll
  - Plugins/WSA/SDK81/x86_64/AVProVideo.dll
  - Plugins/WSA/UWP/ARM/AVProVideo.dll
  - Plugins/WSA/UWP/x86/AVProVideo.dll
  - Plugins/WSA/UWP/x86_64/AVProVideo.dll
  - Plugins/x86/AVProVideo.dll
  - Plugins/x86_64/AVProVideo.dll

4. Usage Notes

4.1 Platform Notes

All graphics APIs are supported:

	D3D9	D3D11	OpenGL	GLES2.0	GLES3.0	Metal
Android	N/A	N/A	N/A	**Yes**	**Yes**	N/A
iOS / tvOS	N/A	N/A	**Yes**	**Yes**	**Yes**	**Yes**
macOS	N/A	N/A	**Yes**	N/A	N/A	**Yes**
Windows	**Yes**	**Yes**	**Yes**	N/A	N/A	N/A
UWP	N/A	**Yes**	N/A	N/A	N/A	N/A

N/A = not applicable

4.1.1 Android

- This plugin requires an API level minimum of 16
- Under the hood we're using the Android MediaPlayer API
- If you want to support streaming don't forget to set the "Internet Access" option in Player Settings to "require"
- For rendering we supports OpenGL ES 2.0 and OpenGL ES 3.0
- Multi-threaded rendering is supported

4.1.2 iOS / tvOS

- Under the hood we're using the AVFoundation API
- If you want to support streaming you need to enable HTTP downloads explicitly. For iOS this is an option in newer versions of Unity, but for Mac OS X and older versions of Unity you have to do this explicitly by editing the plist file. There are notes below on how to do this.
- For rendering we support OpenGL ES 2.0, OpenGL ES 3.0 and Metal
- Multi-threaded rendering is supported

4.1.3 Mac OS X

- Only 64-bit (x86_64) builds are supported (Apple dropped 32-bit support back in 2012 when launching OS X 10.8).
- Under the hood we're using the AVFoundation API
- If you want to support streaming you need to enable HTTP downloads explicitly. For

iOS this is an option in newer versions of Unity, but for Mac OS X and older versions of Unity you have to do this explicitly by editing the plist file. There are notes below on how to do this.

- For rendering on Mac OS X we support OpenGL Legacy and OpenGL Core and Metal.
- Multi-threaded rendering is supported

4.1.4 Windows

- Under the hood we're using the Media Foundation and DirectShow API's. Media Foundation is used for Windows 8 and above while DirectShow is used as a fallback for Windows 7 and below.
- For rendering we support Direct3D 9, Direct3D 11 and OpenGL Legacy
- Multi-threaded rendering is supported

4.1.5 Windows Store / UWP / Hololens

- For streaming video don't forget to enable to "InternetClient" capability option in Unity's Player Settings.

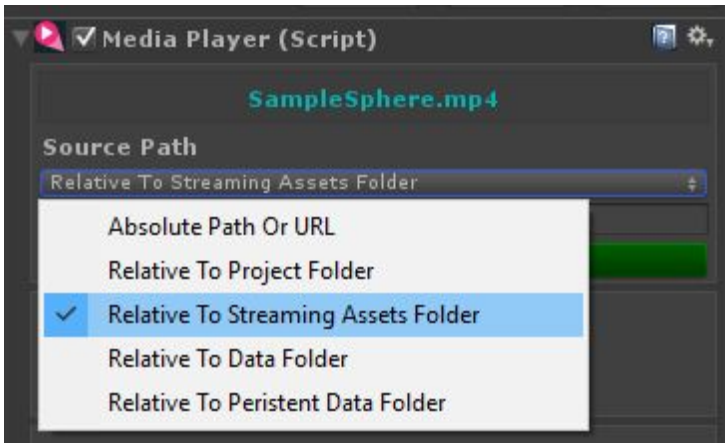
4.1.6 WebGL

- The supported formats and features is dependant on the web browser capabilities.

4.2 Video File Location

Video files can be played in almost any location, however we recommend placing video files in the **/Assets/StreamingAssets/** folder in your Unity project as this is the easiest folder to get started with. StreamingAssets is a special folder that Unity copies to the build without processing. Files copied elsewhere will require manual copying to the build location.

The MediaPlayer component allows you to browse for video files and specify them relative to a parent folder:



The Video Location field specifies the master location of the video file while the Video Path field specifies where to locate the file relative to the Location.

For example if your file is stored in “Assets/StreamingAssets/video.mp4” you would set the Location to “Relative To Streaming Assets Folder” and set the Video Path to “video.mp4”.

Sub-folders are also supported so a video located at “Assets/StreamingAssets/myfolder/video.mp4” would have it’s Video Path set to “myfolder/video.mp4”.

You can also specify absolute paths, URLs or paths relative to other locations:

4.2.1 Relative To StreamingAssets Folder

This is the best and most common location for video files. This folder is located at “Assets/StreamingAssets/” and you must create it if it doesn’t exist. Files copied to this folder will not be imported or processed by Unity but they will be copied with the build automatically.

4.2.2 Absolute Path or URL

Here you can specify a full URL or absolute path to the video file. A URL could be in the form “http://myserver.com/myvideo.mp4” or “rtsp://myserver.com:8080/mystream.rtsp” depending on the platform support and streaming service used.

An absolute path would look like:

- *C:/MyFolder/AnotherFolder/MyVideo.mp4 (Windows)*
- */Users/Mike/downloads/MyVideo.mp4 (Mac/Linux)*
- */Storage/SD/Videos/MyVideo.mp4 (Android external SDCARD)*
- /Storage/emulated/0/MyFolder/MyVideo.mp4 (Android local file system)

Using absolute paths can be useful for testing but isn’t useful when deploying to other machines that don’t necessarily have the same file structure.

4.2.3 Relative To Project Folder

The project folder is the folder of your Unity project, so the folder containing the Assets, Library and Project Settings sub-folders. Specifying files relative to the project folder can be useful when you don't want to include the video files in your Unity Assets folder but want to keep them within the project folder structure. Often making a sub-folder called "Videos" is useful. One possible problem of using this location is that when making a build your video files will not be copied automatically to the build destination so they require manual copying. For builds this folder should be located:

- Windows - at the same level as your EXE
- Mac - at the same level as the Contents folder in your app bundle
- iOS - at the same level as the AppName.app/Data folder
- Android - not accessible due to APK packaging unless you build the APK manually.

4.2.4 Relative To Data Folder

The data folder is specified by Unity here:
http://docs.unity3d.com/ScriptReference/Application-dataPath.html

It isn't that useful to put video files into this folder directly as they would then be processed by Unity into MovieTexture's and will bloat your project size. If you want to stop Unity processing the video files simply rename the extension to something Unity doesn't understand, so "myvideo.mp4" could be renamed to "myvideo.mp4.bin". Files within the data folder (Assets folder in the editor) are not copied automatically to builds so you would have to manually copy them.

4.2.5 Relative to Persistent Data Folder

The persistent data folder is specified by Unity here:
http://docs.unity3d.com/ScriptReference/Application-persistentDataPath.html

For UWP platforms this would resolve to "ms-appdata:///local/"

4.3 Streaming Notes

AVPro Video supports several streaming protocol depending on the platform:

	HTTP Progressive Streaming	HLS	MPEG-Dash	RTSP
Windows Desktop	Yes	Yes (Windows 10 only)	Yes (Windows 10 only)	Only with ASF stream, or with DirectShow using suitable filter
UWP	Yes	Yes (UWP 10 only)	Yes (UWP 10 only)	No
Mac OS X	Yes	Yes	No	No
iOS	Yes	Yes	No	No
tvOS	Yes	Yes	No	No
Android	Yes	Yes, but better on newer versions	No	Yes
WebGL	Yes	Browser specific	Browser specific	No

HTTP Progressive Streaming

When encoding MP4 videos for streaming make sure they are encoded with the video header data at the beginning of the file. You normally do this by selecting “Fast Start” in QuickTime encoder, or use the “-movflags faststart” in FFMPEG, Other encoders will have a similar option. To prepare an MP4 for streaming using FFMPEG you can use the following command:

ffmpeg -i %1 -acodec copy -vcodec copy -movflags faststart %1-streaming.mp4

Vimeo Note:

If you are streaming videos from VIMEO as MP4 then you should note that you can replace the ".mp4" part in the URL with ".m3u8" to instead make it an HLS stream. This may be particularly useful if you are developing apps for the Apple's App Store as you would need to use HLS streaming to pass certification (as for April 2016).

4.3.1 OS X, iOS and tvOS Streaming

This platform supports streaming of HLS streams which typically end with the m3u or m3u8 extension.

If you have an HTTPS URL it should work fine because Apple trusts the secure connection.

If you can only use HTTP then you app has to have a special flag set to let it use HTTP connections (this is a security issue for Apple).

This setting is exposed in the Unity Player Settings here for iOS and tvOS:

The setting is also exposed in the scripting API here:
http://docs.unity3d.com/ScriptReference/PlayerSettings.iOS-allowHTTPDownload.html

If for some reason your version of Unity doesn't expose this then you will have to add it manually. In the Unity editor you need to edit "Unity.app/Contents/Info.plist" and in your built application you would need to edit "your.app/Contents/Info.plist". These files need to have these keys added:

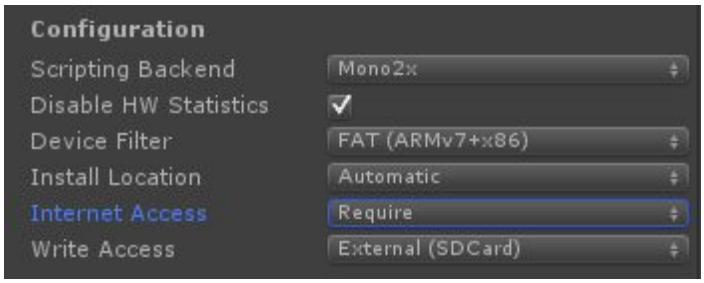
```
<key>NSAppTransportSecurity</key>
<dict>
 <key>NSAllowsArbitraryLoads</key>
 <true/>
</dict>
```

You can find more information about this here:
http://ste.vn/2015/06/10/configuring-app-transport-security-ios-9-osx-10-11/

We've also included a post process build script called "PostProcessBuild.cs" in the project which edits the plist and adds this attribute. Currently it's only set for iOS but you can edit the #define at the top to allow Mac OS X too.

4.3.2 Android Streaming

Android streaming requires the Internet Access setting (in Player Settings) to be set to “Require”:



4.3.3 Test Streams

We found these streams handy for testing (no guarantee that they’re still working):

- **Streaming MP4**
  - http://downloads.renderheads.com/2016/BigBuckBunny_360p30_Streaming.mp4
- **HLS**
  - http://qthttp.apple.com.edgesuite.net/1010qwoeiuryfg/sl.m3u8
  - http://184.72.239.149/vod/mp4:BigBuckBunny_115k.mov/playlist.m3u8
- **MPEG-Dash**
  - http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/RedBullPlayStreets/10sec/RedBull_10_simple_2014_05_09.mpd
- **RTSP**
  - rtsp://rtmp.infomaniak.ch/livecast/latele

4.4 Augmented / Virtual Reality Notes

So far we have tested AVPro Video with:

- Gear VR
- Google Cardboard
- Google Daydream (currently in beta, we’ve seen some performance issues)
- Oculus Rift
- HTC Vive
- Microsoft Hololens

VR is still very new and you should always check for the latest recommended installation steps when creating your project. We found a lot of out of date setup instructions on the net.

AVPro Video supports 4K MP4 playback for creating 360 degree experiences. Stereo 4K

videos in both top-bottom and side-by-side formats are also supported. If you're using Windows 10 and have an Nvidia Geforce 1000 series (eg 1070) you can display 8K H.265 videos!

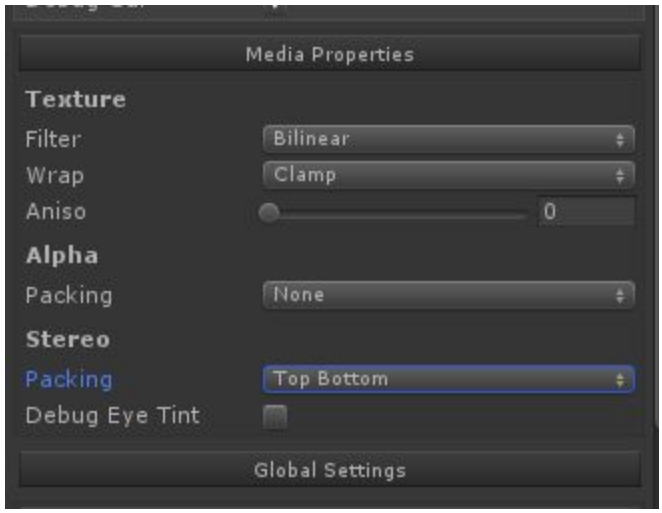
See the FAQ for tips to achieve high resolution video playback for VR.

For software decoders reducing the complexity of the encoded video will give the decoding engine a much easier time and could result in higher frames rates and lower CPU/GPU usage. Possible encoding tweaks include:

- Use the lowest profile level possible
- Don't use too many reference frames
- Don't use too many b-frames
- Disable CABAC

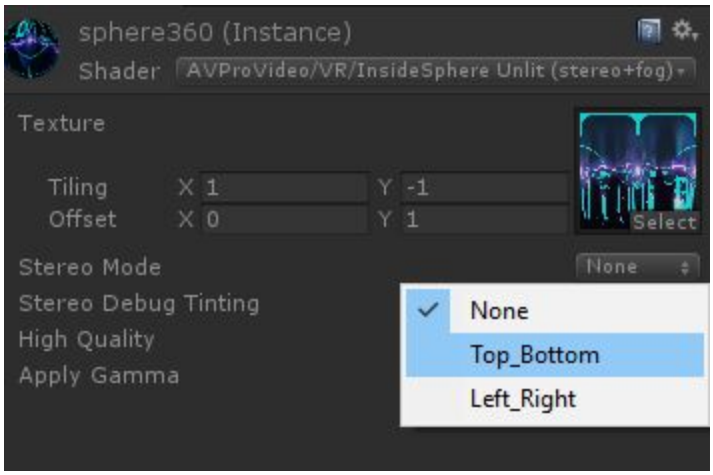
4.4.1 Stereo VR

AVPro Video supports stereoscopic videos in the top-bottom and left-right formats. You can set the stereo packing format of your video in the Media Properties panel:



Now when using the InsideSphere shader on a mesh it will automatically map the right part of the video to each eye. See the "Demo_360SphereVideo" scene for an example of how this works.

Optionally you can manually set the material properties. The included shader "InsideSphere.shader" allows you to easily set what format your video is in via a drop-down in the material:



Select “Stereo Debug Tinting” to colour the left and right eyes different colours so you can be sure the stereo is working.

NOTE: Be sure to add the "UpdateStereoMaterial” component script to your scene when using this material when using Unity 5.3 or below or Unity 5.4 and above without the Single Pass VR option enabled. Often stereo VR requires 2 cameras, each set to a different layer mask and 2 spheres also set to a different mask. AVPro Video doesn’t require this and just uses your normal single camera and single sphere.

4.4.2 VR Audio

Currently only the Windows plugin has support for audio manipulation in VR.
There are two main audio issues with VR in Unity:

1. **Audio redirection**
   Some VR systems such as the Oculus Rift and HTC Vive have their own audio output device and one needs to redirect audio to these devices instead of the system default audio device. Unity does this automatically for its internal audio, but AVPro Video renders to the system default audio device.

   This issue can be solved by either using the AudioOutput component (see below). This component redirects the audio to be rendered by Unity and so it goes to the correct device. AudioOutput requires that the Media Foundation video API be used, so if you need to use the DirectShow API you can specify the name of the output device manually in the field “Force Audio Output Device”:

The device name to use can be retrieved from the VR API or hard coded. For Oculus Rift the name is usually “Rift Audio” and for HTC Vive it is “HTC VIVE USB Audio”.

2. **Audio spatialisation**
   Audio needs to rotate as the user moves their head in the virtual world. This can be achieve by using the AudioOutput component which redirects the audio from the video into Unity (via Unity’s AudioListener).

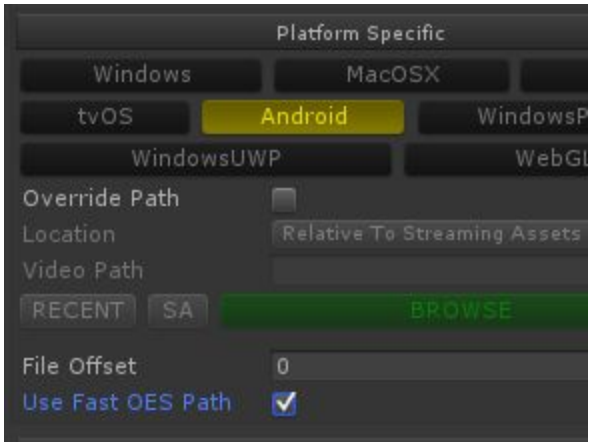
This component should be stacked above its required AudioSource component.

AudioOutput requires that the video API is set to Media Foundation and that the “Use Unity Audio” tickbox is selected.

For proper 3D audio placement you will also need to add a spatialiser plugin to the Unity Audio Settings screen (in Unity 5.4 and above) and tick the “spatialize” tickbox on the Audio Source.

4.4.3 Android OES playback path

For Android there is a special playback option called “Use Fast OES Path”. This option caters especially for VR where users are trying to get the highest possible frame rate and resolution out of the device (without it overheating at the same time). The option is available in the Platform Specific section of the MediaPlayer component:



The OES path is not enabled by default because it requires some special care to be taken and can be tricky for beginners. When this option is enabled the Android GPU returns special OES textures (see EGL extension OES_EGL_image_external) that are hardware specific. Unfortunately Unity isn’t able to use these textures directly, so you can’t just map them to a material or UI. To use the texture a GLSL shader must be used. Unfortunately Unity’s GLSL support isn’t as good as its CG shader support so again this makes things more tricky. The GLSL compiler only happens on the device (not inside Unity) so errors in the shader can be difficult to debug.

We have included a version of the VR sphere shader that supports stereo videos as an example. Hopefully in the future we can improve the integration of these shaders so they aren’t such special cases. This playback path is much faster though, so is definitely worth exploring.

4.5 Hap Codec Notes

The Hap video codec is natively supported by AVPro Video on certain platforms and has the following benefits:

- Very low CPU usage
- GPU decompression
- Low memory usage

- Supports very high resolutions
- Supports alpha channel transparency

The main down side is:

- Very large files

AVI and MOV containers can both be used however we recommend the MOV container. Hap is only supported on Windows and Mac OS X platforms.

4.5.1 Windows Support

Hap, Hap Alpha and HapQ are supported. Hap currently requires the “DirectShow” video API to be selected:

4.5.2 Mac OS X Support

Hap, Hap Alpha and Hap Q are supported.

4.5.3 Encoding

You can download the QuickTime codec for Windows and macOS here:
https://github.com/Vidvox/hap-qt-codec/releases

This codec can be used with QuickTime Pro or any other software that supports QuickTime codecs such as Adobe After Effects and Adobe Premiere.

Alternatively you can use a recent build of FFMPEG with the following command-lines:

- ffmpeg -i input.mov -vcodec hap -format hap output-hap.mov
- ffmpeg -i input.mov -vcodec hap -format hap_alpha output-hap.mov
- ffmpeg -i input.mov -vcodec hap -format hap_q output-hap.mov

Notes:

- Width and height must be multiple of a 4.
- Hap Alpha requires straight not pre-multipled alpha.

4.6 Transparency Notes

Not many video codecs have native support for transparency / alpha channels. Formats supported by some platforms of AVPro Video are:

- Hap Alpha
    - Great support on Windows and Mac OS X. Fast and low overhead format, though file size can get large depending on the content. Currently this is the format we recommend for transparent video.
- Uncompressed RGBA
- Uncompressed YUVA
    - Uncompressed isn't ideal for file size or disk bandwidth but can still be used as a fallback
- ProRes 4444
    - Best support is on Mac OS X. Files are huge.
- VP6
    - Legacy format. We support it only via 3rd party DirectShow plugins for Windows (eg LAV Filters)

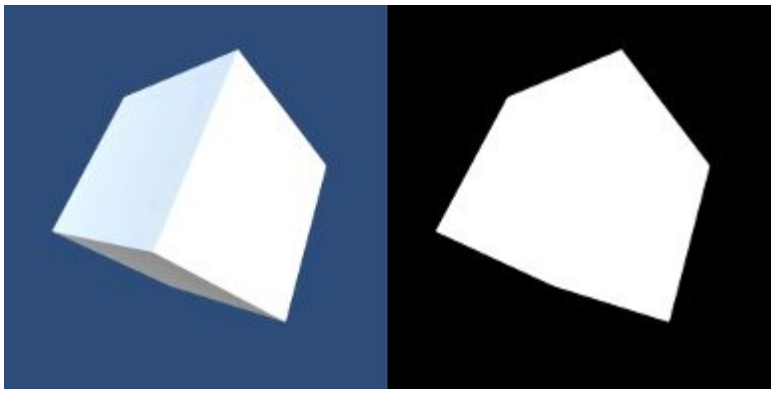
4.6.1 Alpha Packing

Alternatively you can encode your videos in video formats that don't support an alpha channel by packing the alpha channel into the same frame. You can double the width for a left-right packing layout, or double the height for a top-bottom packing layout. This packing could be created in software such as AfterEffects, or the command-line ffmpeg tool can be used.

Here we show two examples using ffmpeg to convert to an alpha packed format:

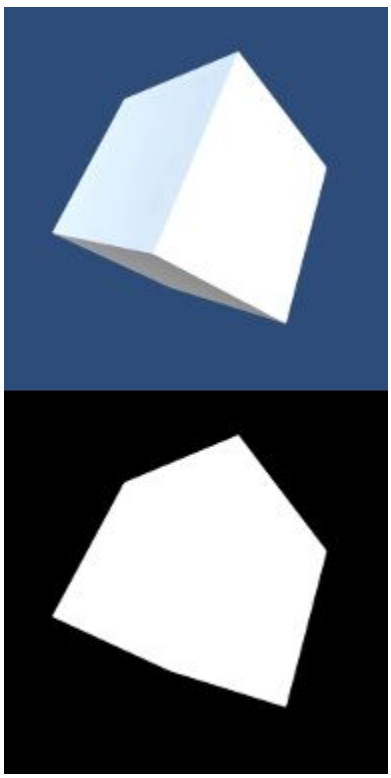
Left-right alpha packing:

```
ffmpeg -i %1 -vf "split [a], pad=iw*2:ih [b], [a] alphaextract, [b] overlay=w" -y %1.mp4
```



Top-bottom alpha packing:

ffmpeg -i %1 -vf "split [a], pad=iw:ih*2 [b], [a] alphaextract, [b] overlay=0:h" -y %1-tb.mp4



4.7 Hardware Decoding

AVPro Video supports hardware decoding on most platforms for optimal performance. WebGL, Windows 8.1, Windows 10, macOS, tvOS, iOS and Android all default to hardware decoding when possible. When playing back multiple videos simultaneously one must be careful not to exceed the number of videos that the hardware can play back smoothly. We have tried to collect information about the limits of specific hardware.

4.7.1 NVidia

NVidia GPU's use a technology called "Purevideo" or "NVDec" to off-load decoding from the CPU. More information can be found here:
https://en.wikipedia.org/wiki/Nvidia_PureVideo#Nvidia_VDPAU_Feature_Sets

Some NVidia Purevideo capabilities:

KEPLER (GK107, GK104)	MAXWELL 1 (GM107, GM204, GM200)	MAXWELL 2 (GM206)	PASCAL (GP100)
MPEG-2, MPEG-4, H.264	MPEG-2, MPEG-4, H.264, HEVC with CUDA acceleration	MPEG-2, MPEG-4, H.264 HEVC/H.265 fully in hardware	MPEG-2, MPEG-4, H.264 HEVC/H.265 fully in hardware
H.264: ~200 fps at 1080p; 1 stream of 4K@30	H.264: ~540 fps at 1080p 4 streams of 4K@30	H.264: ~540 fps at 1080p 4 streams of 4K@30	?
H.265: Not supported	H.265: Not supported	H.265: ~500 fps at 1080p 4 streams of 4K@30	?

(table adapted from presentation "HIGH PERFORMANCE VIDEO ENCODING WITH NVIDIA GPUS")

GPU Architecture	MPEG-2	VC-1	H.264/ AVCHD	H.265/ HEVC	VP8	VP9
Fermi (GF1xx)	Maximum Resolution: 4080x4080	Maximum Resolution: 2048x1024 1024x2048	Maximum Resolution: 4096x4096 Profile: Baseline, Main, High profile up to Level 4.1	Unsupported	Unsupported	Unsupported
Kepler	Maximum	Maximum	Maximum	Unsupported	Unsupported	Unsupported

(GK1xx)	Resolution: 4080x4080	Resolution: 2048x1024 1024x2048	Resolution: 4096x4096 Profile: Main, High profile up to Level 4.1			
Maxwell Gen 1 (GM10x)	Maximum Resolution: 4080x4080	Maximum Resolution: 2048x1024 1024x2048	Maximum Resolution: 4096x4096 Profile: Baseline, Main, High profile up to Level 5.1	Unsupported	Unsupported	Unsupported
Maxwell Gen 2 (GM20x)	Maximum Resolution: 4080x4080	Maximum Resolution: 2048x1024 1024x2048 Max bitrate: 60Mbps	Maximum Resolution: 4096x4096 Profile: Baseline, Main, High profile up to Level 5.1	Unsupported	Maximum Resolution: 4096x4096	Unsupported
Maxwell Gen 2 (GM206)	Maximum Resolution: 4080x4080	Maximum Resolution: 2048x1024 1024x2048 Interlaced	Maximum Resolution: 4096x4096 Profile: Baseline, Main, High profile up to Level 5.1	Maximum Resolution: 4096x2304 Profile: Main profile up to Level 5.1	Maximum Resolution: 4096x4096	Maximum Resolution: 4096x2304 Profile: Profile 0
Pascal (GP100)	Maximum Resolution: 4080x4080	Maximum Resolution: 2048x1024 1024x2048	Maximum Resolution: 4096x4096 Profile: Baseline, Main, High profile up to Level 5.1	Maximum Resolution: 4096x4096 Profile: Main profile up to Level 5.1	Maximum Resolution: 4096x4096	Maximum Resolution: 4096x4096 Profile: Profile 0
Pascal (GP10x)	Maximum Resolution: 4080x4080	Maximum Resolution: 2048x1024 1024x2048	Maximum Resolution: 4096x4096 Profile: Baseline, Main, High profile up to Level 5.1	Maximum Resolution: 8192x8192 Profile: Main profile up to Level 5.1	Supported2 Maximum Resolution: 4096x4096	Maximum Resolution: 8192x8192 Profile: Profile 0

(Table adapted from Nvidia Video Decoder Interface documentation: https://developer.nvidia.com/nvdec-programming-guide)

4.7.2 AMD

AMD UVD
https://en.wikipedia.org/wiki/Unified_Video_Decoder

5. Quick Start Examples

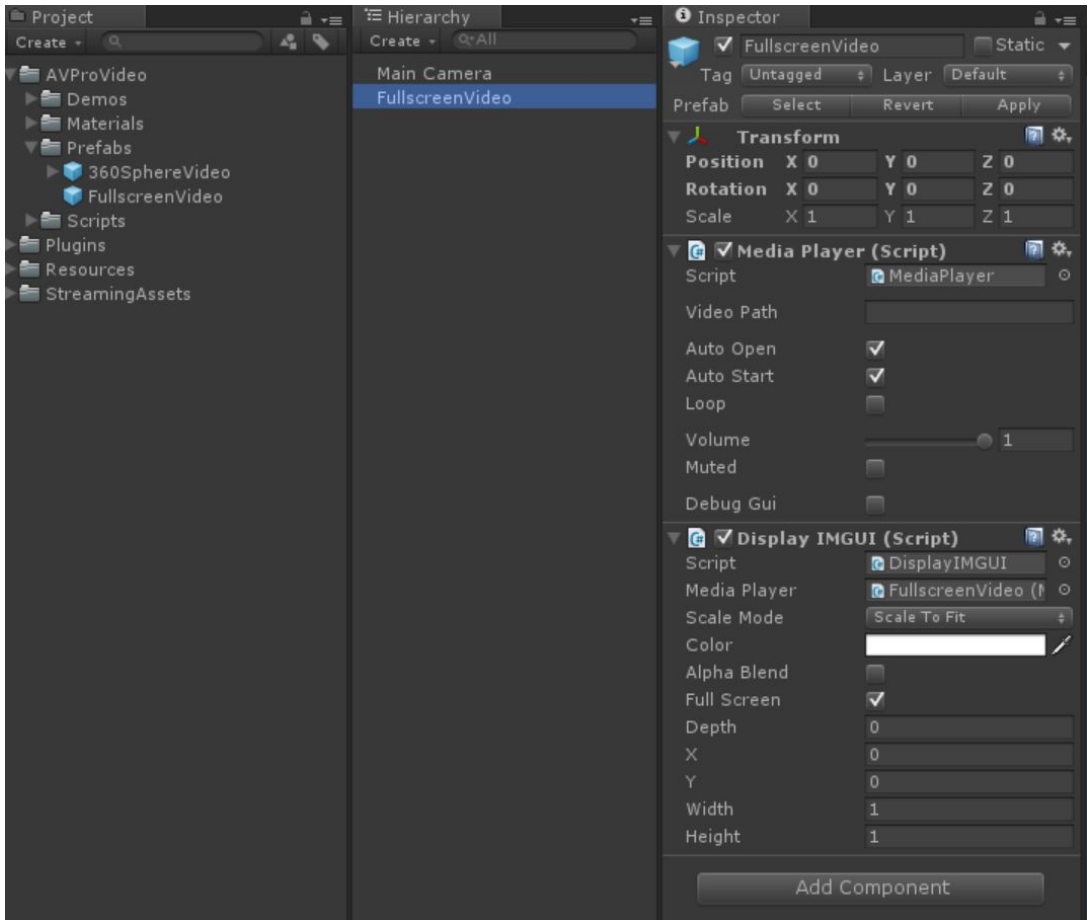
5.1 Quick Start: Fastest Start for Unity Experts

1. Put video files in the StreamingAssets folder
2. Use the MediaPlayer script to play your video (set Video Path to the file name of your video file).
3. Use one of the display scripts to display your video (eg DisplayIMGUI, DisplayUGUI, ApplytoMaterial)

5.2 Quick Start: Fullscreen Video Player using Prefabs

AVPro Video includes a number of example prefabs you can use to easily add video playback to your project. The following steps will create an application that plays back a fullscreen video:

1. Create a new Unity project
2. Import the AVProVideo package
3. From the AVPro/Prefabs folder in the Project window, drag the FullscreenVideo prefab to your Hierarchy window



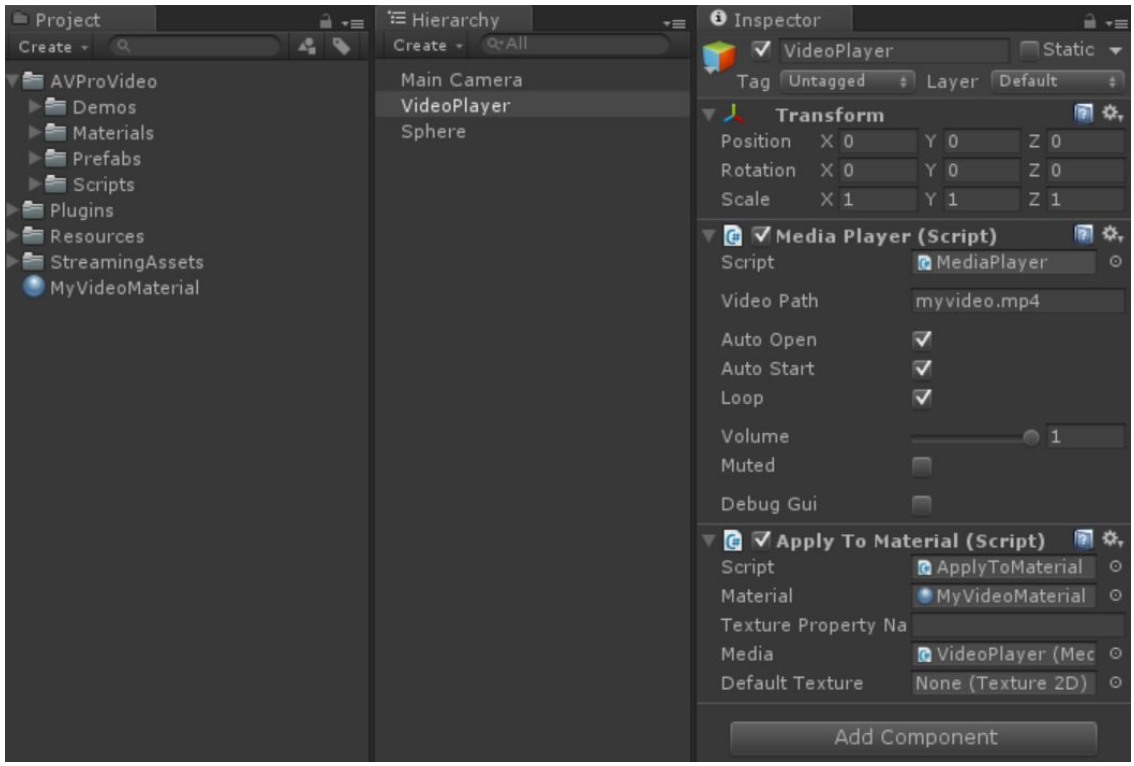
4. Create a folder called StreamingAssets in your Project window and copy your video file (say MP4 file) into that folder
5. Enter the file name (including extension) into the Video Path field in the MediaPlayer component (eg myvideo.mp4)
6. Build and deploy your application, the video will be displayed fullscreen

The DisplayIMGUI component script is just one of the components for displaying video. It uses the legacy Unity IMGUI system which always renders on top of everything else. Try using the DisplayBackground or DisplayUGUI components for more control if you don't want your video to be on top.

5.3 Quick Start: 3D Mesh Video Player using Components

AVPro Video includes a number of easy to use script components you can add to your scene. In this example we show how to use the components to play a video onto a material which is applied to a 3D model in the scene.

1. Create a new Unity project
2. Import the AVProVideo package
3. Create a new GameObject from the "GameObject > AVPro Video > Media Player" menu command
4. Click the "Add Component" button and add "AVPro Video > Apply To Mesh"
5. Drag the Media Player script to the "Media" field in the Apply To Mesh script, this tells the Apply to Mesh script which media player to use
6. Create a sphere via the "GameObject > 3D Object > Sphere" menu command
7. Drag the Mesh Renderer component to the "Mesh" field in the Apply To Mesh script, this tells the Apply to Mesh script which mesh to use



8. Create a folder called StreamingAssets in your Project window and copy your video file (say MP4 file) into that folder
9. Enter the file name (including extension) into the Video Path field in the MediaPlayer component (eg myvideo.mp4)
10. Build and deploy your application, the video will be displayed on your 3D sphere

6. Usage

6.1 Getting Started

The easiest way to get started is to look at the included demos and see what script components have been used. For video playback you need 3 things in your scene:

1. The video file to play:

   Create a “StreamingAssets” folder in your Project window
   Copy your video file (usually MP4 file, but consult the list of supported formats for your platform below) to the StreamingAssets folder

2. A MediaPlayer script to load and play the video:

   Create a GameObject and add the MediaPlayer script to it
   Set the Video Path field to the name of your video file (e.g. myvideo.mp4)

3. A script to display the video:

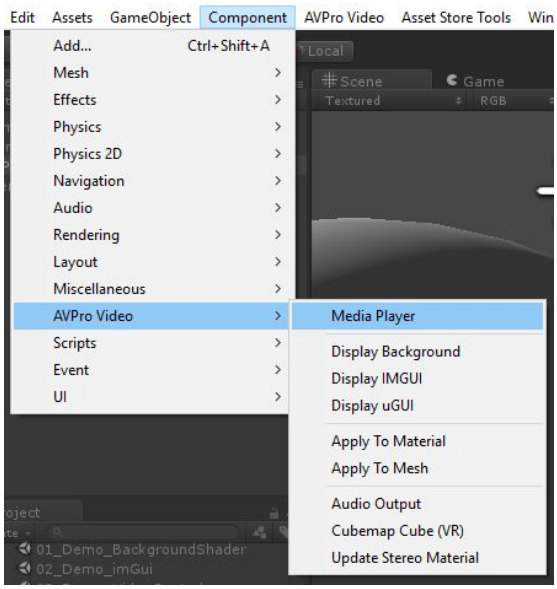
   Decide how and where you want your video file to appear. There are a number of different display component scripts included for different usage scenarios. If you want to display the video on top of everything in your scene just add the DisplayIMGUI script to a GameObject in your scene and set the Media Player field your MediaPlayer component. Other display components work similarly.

6.2 Unsupported Platform Fallback

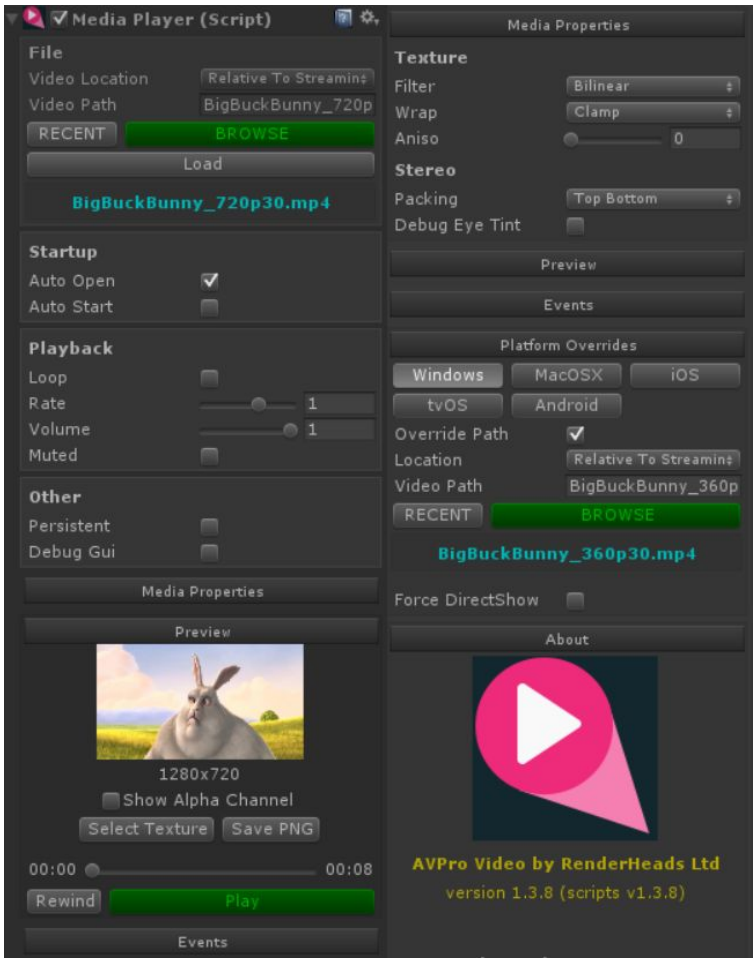
AVPro Video is designed to still function even on platforms that aren’t natively supported. Instead of displaying the actual video though, a dummy 10 second long “AVPro” visual is shown. All of the video controls should still work. For example if you are running your editor in Linux the dummy video player will appear in the editor and the real video will appear when you deploy to supported platforms. If you deploy to an unsupported platform such as Samsung TV you will also see the dummy video player. The code is easily extendable to add custom video players for any unsupported platform.

6.3. Components

Included are a number of components to make this asset easy to use. The components are located in the AVProVideo/Scripts/Components folder or you can add them from the Components menu:



6.3.1 Media Player Component



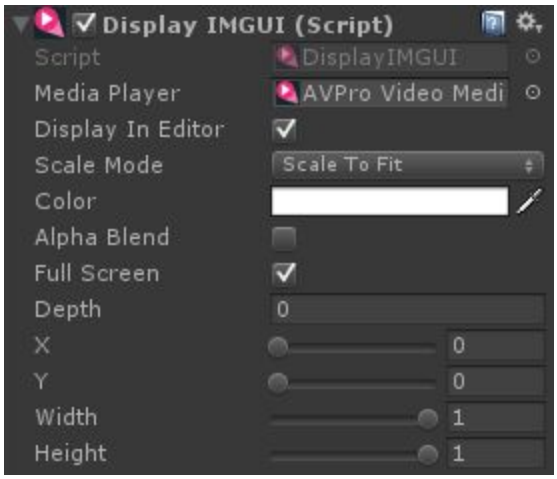
This is the core component for playing media. This component only handles the loading and playback of media and doesn't handle how it is displayed. Use the display script

components to control how and where the video is displayed. Fields are:

- Video Location
    - Where to look for the file specified in the Video Path below. This can be an absolute path/URL, or relative to one of the Unity folders. The StreamingAssets folder is the easiest to use. Options are:
        - Absolute or URL
            - This is an absolute path on your device, or an http URL
        - Relative to Project Folder
            - The root folder is the folder above your Assets folder
        - Relative to Streaming Assets Folder
            - The root folder is /Assets/StreamingAssets
        - Relative to Data Folder
            - The root folder is /Assets
            - Unity manual has more information: http://docs.unity3d.com/ScriptReference/Application-dataPath.html
        - Relative to Persistent Data Folder
            - Unity manual has more information: http://docs.unity3d.com/ScriptReference/Application-persistentDataPath.html
- Video Path
    - The file path to the video in the StreamingAssets folder (e.g. myvideo.mp4 or AndroidVideos/myvideo.mp4 if you want to use a subfolder)
- Auto Open
    - Whether to open the file when this component is enabled/starts
- Auto Start
    - Whether to play the video once a video is opened
- Loop
    - Whether to loop the video
- Playback Rate
    - Sets a multiplier that affects video playback speed
    - Not supported on Android
- Volume
    - 0..1 range for audio volume
- Muted
    - Whether the audio is muted
- Persistent
    - Applies DontDestroyOnLoad to the object so that it survives scene/level loads
- Debug Gui
    - Whether to display an overlay with statistics on the video playback - useful for debugging
- Events
    - This event can be hooked up to scripting functions which will get called when a non-looping video completes playback. See the Events section below for more details and a scripting example

- Platform overrides
    - These allow you to set a different file per platform.

6.3.2 Display IMGUI Component

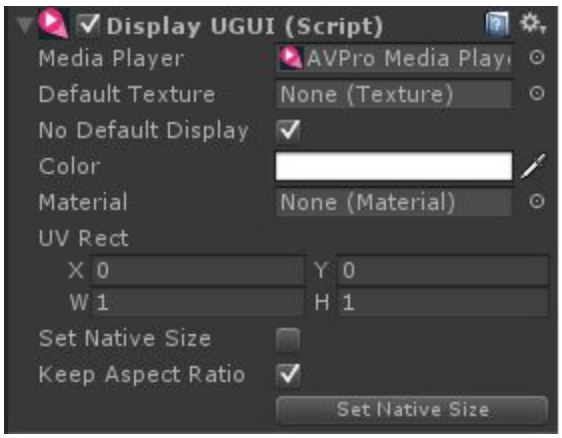


This is the most basic component for displaying the video. It uses the legacy Unity IMGUI system to display the video to the screen. IMGUI is always rendered on top of everything else in the scene, so if you require your video to be rendered in 3D space or as part of the uGUI system it's better to use the other components. Fields are:

- Media Player
    - The media player to display
- Display in Editor
    - Whether to display the rectangle in the editor - useful for debugging
- Scale Mode
    - How to fit the video to the screen
- Color
    - The color to tint the video, including alpha transparency
- Alpha Blend
    - Whether the video texture controls transparency. Leaving this off for opaque videos is a minor optimisation
- Depth
    - The IMGUI depth to display at. Use this to change the order of rendering with other IMGUI scripts
- Full Screen
    - Whether to ignore the X, Y, Width, Height values and just use the whole screen
- X
    - The normalised (0..1) x position
- Y
    - The normalised (0..1) y position

- Width
    - The normalised (0..1) width
- Height
    - The normalised (0..1) height

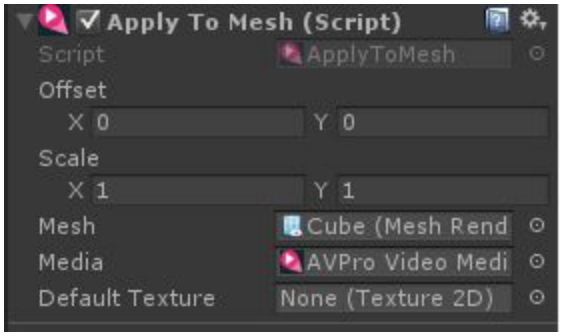
6.3.3 Display uGUI Component



This component is used to display a video using Unity's uGUI system. Field are:

- Media Player
    - The media player to display
- Default Texture (optional)
    - A texture to display while the video isn't playing (while it is buffering for example).
- No Default Display
    - Will not show anything until there are frames available
- Color
    - The color to tint, including alpha transparency
- Material
    - Standard uGUI field, change this to one of the included materials when using packed alpha or stereo videos
- UV Rect
    - Standard uGUI field
- Set Native Size
    - When the video loads will resize the RectTransform to the pixel dimensions of the video
- Keep Aspect Ratio
    - Whether to keep the correct aspect ratio or stretch to fill

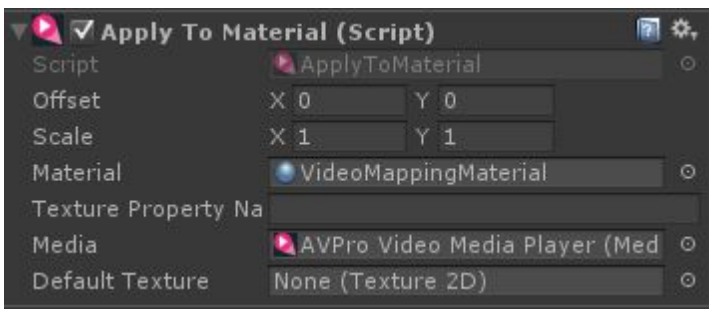
6.3.4 Apply To Mesh Component



This component takes the texture generated by the Media Player component and assigs it to the texture slot of the material on a 3D Mesh. This is useful for playing videos on 3D meshes. Field are:

- Offset
    - The XY translation to apply to the texture
- Scale
    - The XY scale to apply to the texture
- Mesh
    - The mesh (renderer) to apply the texture to
- Media
    - The media player
- Default Texture (optional)
    - A texture to display while the video isn't playing (while it is buffering for example).

6.3.5 Apply To Material Component

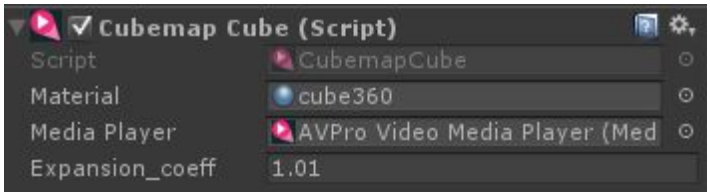


This component takes the texture generated by the Media Player component and assigns it to a texture slot in a Material. This is useful for playing videos on 3D meshes. Fields are:

- Offset
    - The XY translation to apply to the texture
- Scale
    - The XY scale to apply to the texture
- Material

    - The material to apply the video texture to
- Texture Property Name (optional)
    - By default this script assigns to the main texture (_MainTex) but if you want to assign to another slot you can put the name in here
- Media
    - The media player
- Default Texture (optional)
    - A texture to display while the video isn't playing (while it is buffering for example).

6.3.6 Cubemap Cube Component



This component generates a cube mesh that is suitable for 3:2 cubemap 360 VR videos. Fields are:

- Material
    - The material to apply to the cube. This is usually just a standard unlit material.
- Media Player
    - The media player that will have its video texture displayed on the cube
- Expansion_coeff
    - The value used during enabling to pad the edges to the video to prevent bilinear bleed artifacts. Default is 1.01.

6.3.7 Audio Output Component

This component currently only supports the Windows platforms (Windows 8 and above) and is used to pass Audio from the Media Player to Unity. This allows audio effects,3D placement and 360 VR spatialisers to be used. Fields are:

- Media Player
    - The media player that will have its audio outputted via Unity

6.4 Scripting

6.4.1 Namespace

All scripts use the namespace RenderHeads.Media.AVProVideo so be sure to add “using RenderHeads.Media.AVProVideo” to the top of your source files.

6.4.2 Media Player Scripting

Most scripting is likely to center around the MediaPlayer.cs script. This script handles the loading, playback and updating of videos. The script exposes a number of interfaces related to different use cases and can be found in Interfaces.cs

MediaPlayer exposes 3 main interfaces:

- Info Interface
    - The IMediaInfo interface is exposed by the Info property
    - This interface is used to access information about the media, eg:

```
MediaPlayer mp;
mp.Info.GetVideoWidth();
```

- Control Interface
    - The IMediaControl interface is exposed by the Control property
    - This interface is used to control playback, eg:

```
MediaPlayer mp;
mp.Control.Pause();
```

- TextureProducer interface
    - The IMediaProducer interface is exposed by the TextureProducer property
    - This interface is used to get information about how to display the current texture and is used by the Display components, eg:

```
MediaPlayer mp;
Texture videoTexture = mp.TextureProducer.GetTexture();
```

The MediaPlayer script also has a number of methods for controlling loading of media:

- OpenVideoFromFile()
    - Loads the video specified. Useful if you need manual control over when the video is loaded
- CloseVideo()
    - Closes the video, freeing memory

6.4.3 Events

MediaPlayer currently has these events:

- MetaDataReady - Called when the width, height, duration etc data is available
- ReadyToPlay - Called when the video is loaded and ready to play
- Started - Called when the playback starts
- FirstFrameReady - Called when the first frame has been rendered
- FinishedPlaying - Called when a non-looping video has finished playing
- Closing - Called when the the media is closing
- Error - Called when an error occurred, usually during loading

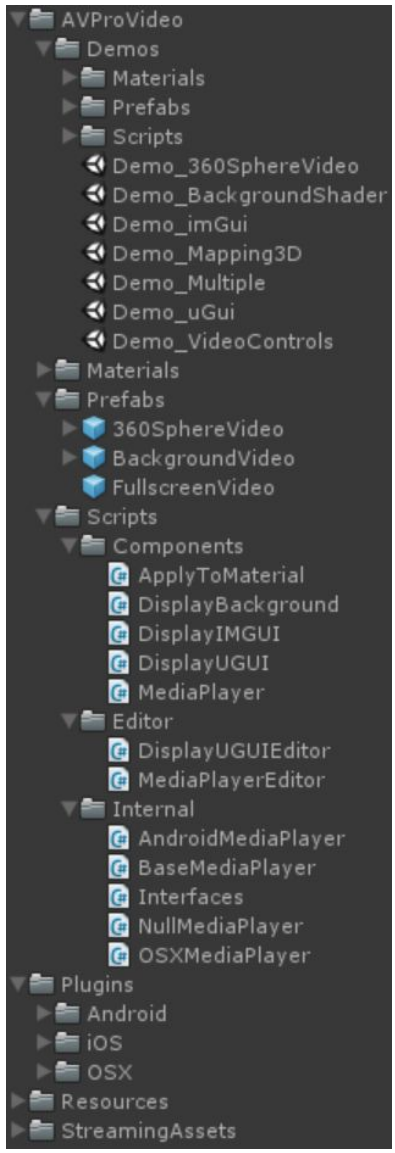
Scripting example:

```
// Add the event listener (can also do this via the editor GUI)
MediaPlayer mp;
mp.Events.AddListener(OnVideoEvent);

// Callback function to handle events
public void OnVideoEvent(MediaPlayer mp, MediaPlayerEvent.EventType et,
ErrorCode errorCode)
{
	switch (et)
	{
		case MediaPlayerEvent.EventType.ReadyToPlay:
		mp.Control.Play();
		break;
		case MediaPlayerEvent.EventType.FirstFrameReady:
		Debug.Log("First frame ready");
		break;
		case MediaPlayerEvent.EventType.FinishedPlaying:
		mp.Control.Rewind();
		break;
	}

	Debug.Log("Event: " + et.ToString());
}
```

7. Asset Files



7.1 Demos

- Demo_360SphereVideo.unity
    - Demo contains a video player that plays a 360 degree video using equirectangular(lat-long) mapping.
    - The video is applied to a sphere, inside of which is the main camera.
    - If the target device has a gyroscope then moving the device around with rotate the camera to view the video from different angles. For platforms without gyroscope the mouse/touch can be used to look around.
    - A special shader and script are used to allow a single camera to render in stereo on a VR headset. Click on the material to set whether it should display the video as monoscopic, stereo top-bottom or stereo left-right.
- Demo_360CubeVideo.unity
    - Same as the sphere demo above, but using a cubemap 3x2 layout source video.
- Demo_BackgroundShader.unity
    - Basic demo that plays a video using the background material which allows the video to appear behind all content.
- Demo_FrameExtract.unity
    - Shows go to read frames out of the video for saving to disk (jpg/png) or accessing pixel data.
- Demo_imGui.unity
    - Basic demo that plays a video and uses the legacy IMGUI display component to draw the video to the screen.
    - Also has an audio clip to show audio-only media playback.
    - Also has 3 different streaming URLs to demonstrate streaming.
    - IMGUI is drawn on top of all other visual components.
- Demo_Mapping3D.unity
    - Demo containing a video player and a 3D scene
    - Some of the 3D models have the video mapped to them via the ApplyToMaterial script
- Demo_Multiple.unity
    - This demo allows you to programmatically multiple load videos and test multiple videos playing at once. Display is via the AVPro Video uGUI component
- Demo_uGUI.unity
    - This demo shows how to display videos within the uGUI system. It uses the

DisplayUGUI component in the canvas hierarchy.
    - It also uses a custom shader to overlay text with a video texture.
- Demo_VideoControl.unity
    - This demo shows how to query the video state and control playback

7.2 Prefabs

- 360SphereVideo.prefab
    - Prefab containing a video player and mapping to a sphere. Useful for playback of equirectangular 360 degree videos
- BackgroundVideo.prefab
    - Prefab containing a video player and a quad model with a special background material applied. This material makes the quad get drawn before everything else so it appears in the background.
- FullscreenVideo.prefab
    - Prefab controls a video player and the IMGUI display component for very easy basic video playback creation

7.3 Scripts

- Components
    - ApplyToMaterial.cs
        - Applies the texture produced by the MediaPlayer component to a unity material texture slot
    - ApplyToMesh.cs
        - Applies the texture produced by the MediaPlayer component to a Untiy mesh (via MeshRenderer) by setting the mainTexture field of all its materials
    - CubemapCube.cs
        - Generates a cube mesh that can be used for displaying a 3:2 cubemap packed video
    - DisplayBackground.cs
        - Displays the texture produced by the MediaPlayer component behind all other content (not compatible with SkyBox)
    - DisplayIMGUI.cs
        - Displays the texture produced by the MediaPlayer component using Unity's legacy IMGUI system
    - DisplayUGUI.cs
        - Displays the texture produced by the MediaPlayer component using Unity's new uGUI system
    - MediaPlayer.cs
        - The main script for loading and controlling an instance of video playback
    - UpdateStereoMaterial.cs
        - A helper script for VR stereo rendering to update the camera position variable in a spherical material to help work out which eye to render

    - AudioOutput.cs
        - Used to play audio from the media vis Unity's sound system (currently Windows only)
    - ApplyToTextureWidgetNGUI.cs
        - Applies the texture produced by the MediaPlayer component to an NGUI Texture widget texture slot
- Editor
    - DisplayUGUIEditor.cs
        - The editor script that controls how the DisplayUGUI component is rendered in the Inspector
    - MediaPlayerEditor.cs
        - The editor script that controls of the MediaPlayer component is rendered in the Inspector
- Internal
    - AndroidMediaPlayer.cs
        - Android specific media player
    - BaseMediaPlayer.cs
        - Common base class for all platform media players
    - Interfaces.cs
        - Interfaces and events
    - NullMediaPlayer.cs
        - The fallback dummy media player for unsupported platforms
    - OSXMediaPlayer.cs
        - iOS and OSX specific media player
    - WebGLMediaPlayer.cs
        - WebGL specific media player
    - WindowsMediaPlayer.cs
        - Windows specific media player

8. Scripting Reference

AVPro Video is designed to be used mainly with the supplied drag and drop component but there are always times when a bit of scripting is needed. The asset includes sample scenes which give some examples of how to use scripting to control video playback, apply video textures to materials etc which are useful to learn from. The full class reference is available online here:
http://downloads.renderheads.com/docs/AVProVideoClassReference/

In this document we have included a simplified version of the highlights.

MediaPlayer class

The MediaPlayer class is the main class for video playback and is where video files are specified and controlled. This class is mainly controlled via the Unity Inspector UI and for scripting through the interface properties it exposes.

Properties

- Events
  - returns the MediaPlayerEvent class
- Info
  - returns the IMediaInfo interface
- Control
  - returns the IMediaControl interface
- TextureProducer
  - returns the IMediaProducer interface

Methods

All of these methods use the interfaces exposed above and are just handy shortcuts

- void OpenVideoFromFile(FileLocation location, string path, bool autoPlay)
  - Opens the video specified
- void CloseVideo()
  - Closes the current video and frees up allocated memory
- void Play()
  - Starts playback of the video
- void Pause()
  - Pauses the video
- void Stop()
  - Pauses the video
- void Rewind(bool pause)
  - Rewinds the video with an option to pause it as well
- Texture2D ExtractFrame(Texture2D target, float timeSeconds, int timeoutMs)

    - Extracts a frame from the specified time of the current video as a readable Texture2D. This can then be used to save out the pixel data. The texture must be destroyed by the user. The texture can be passed in again via the "target" parameter to reuse it.

IMediaInfo interface

This interface is used to query properties of the video

Methods

- float GetDurationMs();
    - Returns the duration of the video in milliseconds
- int GetVideoWidth();
    - Returns the width of the video in pixels
- int GetVideoHeight();
    - Returns the height of the video in pixels
- float GetVideoFrameRate();
    - Returns the frame rate of the video in frames per second
- float GetVideoDisplayRate();
    - Returns the actual frame rate achieved by the video decoder
- bool HasVideo();
    - Returns whether the media has visual tracks
- bool HasAudio();
    - Returns whether the media has audio tracks
- int GetAudioTrackCount();
    - Returns the number of audio tracks
- string GetPlayerDescription();
    - Returns a string describing the internal playback mechanism

IMediaControl interface

This interface is used to control loading and playback of the video

Methods

- bool OpenVideoFromFile(string path);
    - Starts loading the file from the specified path or URL. Returns false if any error was encountered. This function is asynchronous so the video properties will not be available immediately. This function shouldn't be used, instead use the MediaPlayer OpenVideoFromFile function.
- void CloseVideo();
    - Closes the video and any resources allocated
- void SetLooping(bool looping);

    - Sets whether the playback should loop or not. This can be changed while the video is playing.
- bool CanPlay();
    - Returns whether the video is in a playback state. Sometimes videos can take a few frames before they are ready to play.
- void Play();
    - Starts playback of the video
- void Pause();
    - Pause the video
- void Stop();
    - Stops the video (essentially the same as Pause)
- bool IsPlaying();
    - Returns whether the video is currently playing
- bool IsPaused();
    - Returns whether the video is currently paused
- bool IsFinished();
    - Returns whether the video has completed playback
- bool IsLooping();
    - Returns whether the video has been set to loop
- bool IsBuffering();
    - Returns whether a streaming video has stopped and is buffering. A buffering video will resume after it has downloaded enough data.
- void Rewind();
    - Sets the current time to the beginning of the video
- void Seek(float timeMs);
    - Sets the current time to a specified value in milliseconds
- void SeekFast(float timeMs);
    - Sets the current time to a specified value in milliseconds but sacrifices accuracy for speed. This is useful if you just want to jump forward/back in a video but you don't care about the accuracy.
- bool IsSeeking();
    - Returns whether the video is currently seeking. During seeking no new frames are produced.
- float GetCurrentTimeMs();
    - Returns the current time (playback position) in milliseconds
- void SetPlaybackRate(float rate);
    - Sets the current playback rate. 1.0f is normal rate. Negative rates aren't supported on all platforms.
- float GetPlaybackRate()
    - Returns the current playback rate
- void MuteAudio(bool mute)
    - Sets the audio mute or not
- void SetVolume(float volume)
    - Sets the volume between 0.0 and 1.0
- float GetVolume()
    - Returns the volume level between 0.0 and 1.0

- int GetCurrentAudioTrack()
    - Returns the index of the currently enabled audio track
- void SetAudioTrack(int index)
    - Sets the index to select the audio track to enable exclusively
- float GetBufferingProgress()
    - Returns a value between 0.0 and 1.0 representing network buffering
- ErrorCode GetLastError()
    - Returns an error code is an error occurred this frame

IMediaProducer interface

Methods

- Texture GetTexture();
    - Returns a Unity Texture object if there is a texture available otherwise null is returned.
- int GetTextureFrameCount();
    - Returns the number of times the texture has been updated by the plugin. This can be useful to know when the texture was updated as the value will increment each time.
- long GetTextureTimeStamp();
    - Returns the presentation time stamp of the current texture in 100-nanosecond units. This is useful for accurate frame syncing.
- bool RequiresVerticalFlip();
    - Some textures are decoded up-side-down and need to be vertically flipped when displayed. This method returns whether the texture needs to be flipped during display.

9. Supported Media Formats

In general the most common format that is supported are MP4 files with H.264 encoding for video and AAC encoding for audio. This format is supported across all platforms though not necessarily all bit-rates and profiles.

Container support:

	Windows Desktop	Mac OS X Desktop	iOS, tvOS	Android
MOV	Yes	Yes	Yes	No
MP4	Yes	Yes	Yes	Yes
AVI	Yes	No	No	?
MKV	Yes in Windows 10	?	?	Yes
Webm	No	No	No	Yes
ASF/WMV	Yes	No	No	No
MP3	Yes	Yes	Yes	Yes
WAV	Yes	?	?	?

Audio Codec support:

	Windows Desktop	Mac OS X Desktop	iOS, tvOS	Android
MP3	Yes	Yes	Yes	Yes
FLAC	Yes in Windows 10	No	No	Yes
AAC	Yes	Yes	Yes	Yes
AC3	Yes	Yes	?	?
WMA	Yes	No	No	No
MIDI	Yes	?	?	?
ALAC (Apple Lossless)	No	Yes	Yes	No

µLAW	Yes	Yes	Yes	No
ADPCM	Yes	Yes	Yes	No
Linear PCM	Yes	Yes	Yes	Yss

Video Codec support:

	Windows Desktop	**Mac OS X Desktop**	**iOS, tvOS**	**Android**
HEVC / H.265	Yes in Windows 10	Not yet**	Not yet**	Yes
H.264	Yes	Yes	Yes	Yes
H.263	Yes	?	?	Yes
MJPEG	Yes	No	No	No
WMV	Yes	No	No	No
VP8	Yes with codec*	No	No	Yes
VP9	Yes with codec*	No	No	Yes
Hap	Yes***	Yes	No	No
Hap Alpha	Yes***	Yes	No	No
Hap Q	Yes***	Yes	No	No
ProRes 422	No	Yes	No	No
ProRes 4444	No	Yes	No	No
DV	Yes	Yes in Yosemite and above	No	No
Uncompressed RGBA	Yes	?	?	?
Uncompressed R10K	No	Yes in Yosemite and above	No	No
Uncompressed V210	?	Yes in Yosemite and above	No	No

Uncompressed 2VUY	?	Yes in Yosemite and above	No	No

* The VP8/VP9 Windows Media Foundation codec can be downloaded at http://www.webmproject.org/ie/
** Apparently Apple will release support soon
*** Requires option “Force DirectShow” to be ticked

Cells with “?” are one’s we’re not sure about. We will do more testing and continue to update this table. For more details on which codecs and what type of encoding is best, see the per-platform details below.

9.1 Android

Android supports many media formats. For a complete list check the Android documentation here: http://developer.android.com/intl/ko/guide/appendix/media-formats.html

HEVC support was officially added in Android 5.0 (Lollipop) but mostly as a software decoding implementation.

A list of media-player related Android chipsets and which formats they support for hardware decoding: http://kodi.wiki/view/Android_hardware

9.2 iOS, tvOS and OS X

Many media formats are supported on iOS including H.264. For a complete list check the iOS documentation here: https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/MediaLayer/MediaLayer.html

In OS X, ProRes 422 and ProRes 4444 are supported.

OS X Yosemite and above, supports the following additional formats:

- DV
- Uncompressed R10k
- Uncompressed v210
- Uncompressed 2vuy

9.3 Windows

A full list of supported formats can be found here: https://msdn.microsoft.com/en-us/library/windows/desktop/dd757927(v=vs.85).aspx

https://msdn.microsoft.com/en-us/windows/uwp/audio-video-camera/supported-codecs

H.264 decoder supports up to profile L5.1, but Windows 10 supports above L5.1 profile:
https://msdn.microsoft.com/en-us/library/windows/desktop/dd797815(v=vs.85).aspx

H.265 decoder specs are here:
https://msdn.microsoft.com/en-us/library/windows/desktop/mt218785(v=vs.85).aspx

Windows 10 adds native support for the following formats:

- H.265 / HEVC
- MKV
- FLAC
- HLS Adaptive Streaming

9.4 Windows Phone / UWP

Details on media supported by this platform can be found is platform are here:
https://msdn.microsoft.com/library/windows/apps/ff462087(v=vs.105).aspx
https://msdn.microsoft.com/en-us/windows/uwp/audio-video-camera/supported-codecs

9.5 WebGL

Support for WebGL platform is still varied and depends on the platform and browser support. Some formats such as AVI file container are not supported at all. As with all other platforms, H.264 video in an MP4 container is the most widely supported format.

Adaptive streaming (such as HLS) is still not supported natively by most browsers, but we have seen it working in the Microsoft Edge and Safari browsers.

Here are some resources about the supported formats:

https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats
https://en.wikipedia.org/wiki/HTML5_video#Browser_support
http://www.encoding.com/html5/

10. Support

If you are in need of support or have any comments/suggestions regarding this product please contact us.

Email: unitysupport@renderheads.com
Website: http://renderheads.com/product/avpro-video/
Unity Forum:
http://forum.unity3d.com/threads/released-avpro-video-complete-video-playback-solution.385611/

10.1 Bug Reporting

If you are reporting a bug, please include any relevant files and details so that we may remedy the problem as fast as possible.

Essential details:

- Error message
  - The exact error message
  - The console/output log if possible
  - If it's an Android build then an "adb logcat" capture
- Hardware
  - Phone / tablet / device type and OS version
- Development environment
  - Unity version
  - Development OS version
  - AVPro Video plugin version
- Video details:
  - Resolution
  - Codec
  - Frame rate
  - Better still, include a link to the video file

Better still, send us a full or reduced copy of your Unity project

11. About RenderHeads Ltd

RenderHeads Ltd is an award winning creative and technical company that has been designing and building cutting edge technology solutions since its formation in 2006. We specialise in creating interactive audio-visual software for installations at auto shows, museums, shows and expos.

11.1 Services

- Unity plugin development
- Unity game / interaction / virtual and augmented reality development
- Unity consulting

11.2 Our Unity Plugins

Many of the apps and projects we develop require features that Unity doesn't yet provide, so we have created several tools and plugins to extend Unity which are now available on the Unity Asset Store. They all include a **free trial or demo version** that you can download directly from the website here:
http://renderheads.com/product-category/for-developers/

11.2.1 AVPro Video

Powerful cross-platform video playback solution for Unity, featuring support for Windows, OS X, iOS, Android and tvOS. This is our newest plugin.

11.2.2 AVPro Movie Capture

Video capture to AVI files direct from the GPU and encoded to files using DirectShow codecs. Features include 4K captures, lat-long (equirectangular) 360 degree captures, off-line rendering and more. Windows only.

11.2.3 AVPro Live Camera

Exposes high-end webcams, tv cards and video capture boards to Unity via DirectShow. Windows only.

11.2.4 AVPro DeckLink

Integrates DeckLink capture card functionality into Unity, allowing users to send and receive high-definition uncompressed video data to and from these capture cards.

11.2.5 Screenshot Annotator Pro

Highly productive tool allowing in-game and in-editor annotation of screenshots which can then be shared with your team via FTP, Email, Slack or Teamwork.com with the click of the mouse. Cross-platform.

Appendix A - Frequently Asked Questions

1. **Why won't my high-resolution video file play on Windows?**

   The ability to play back high resolution videos depends on the version of Windows operating system and which video codecs you have installed.

   By default AVPro Video will try to use the standard Microsoft codecs that come with Windows. These have limitations, for example:

Decoder	Windows 7 and below	Windows 8+
h.264	**1080p**	2160p (4K)

   If you want to use 3rd party codecs you can install them to improve the range of resolution and format options available. These come in the form of Media Foundation or DirectShow codecs. The LAV Filters are a great example of DirectShow codecs that will allow for higher resolution video decoding.

2. **Does Time.timeScale affect video playback speed?**

   Yes we have BETA support for time.timeScale and time.captureFramerate. Simply enable the option on the Media Player component in the panel labelled "Global Settings".

3. **Does AVPro Video support playing YouTube videos or live streams?**

   Yes and no. If you enter a YouTube URL into AVPro Video it will not be able to play it because this is the URL to the website and not the video. It is possible to gain access to the underlying MP4 file or HLS stream URL which will work with AVPro Video. This may be against the terms and conditions of YouTube though.

4. **How can I get smoother video playback for my Windows VR app?**

   This is usually caused by the video decoding taking too much CPU time. Some videos (especially highly compressed ones) require a lot of CPU time to decode each frame.

   Try enabling the hardware decoding option in "Platform Specific" panel. This will enable your application to use GPU decoding which is much faster. This option is only supported on Windows 8.1 and above and when D3D11 graphics API is used.

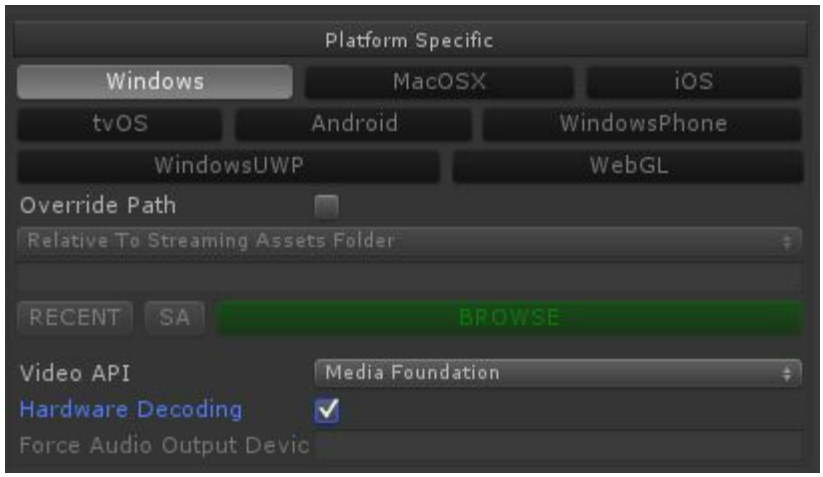
   You could also try re-encoding your video using settings that are less demanding for

the decoder. Most H.264 encoders have a 'fast decode' preset, or you can manually change settings such as disabling CABAC, reducing the number of reference frames, reducing bitrate, disabling B-frames, disabling the loop (deblocking) filter etc.

You could also try switching to another video codec that allows for less CPU intensive decoding. Codecs such as H.264 / H.265 are generally very heavy on the CPU. Hap is an extremely fast codec but will result in large files.

5. **Is GPU hardware decoding available?**

Yes it is on most platforms. Android, iOS, tvOS and macOS mostly use GPU decoding - but it depends on the codec. For Windows GPU decoding is enabled by default but is only available for Windows 8.1 and above and when using D3D11 graphics API. You can toggle GPU decoding is enabled via the Platform Specific panel:



6. **Is multi-channel audio supported?**

Audio with more than 2 channels should be supported on desktop platforms. On Windows you will need to set your sound settings in the Control Panel for the number of channels you want to output. If you leave your Windows sound settings as stereo then the audio will get mixed to stereo.

7. **Why isn't seeking accurate / responsive?**

The way seeking works depends on the platform. Some platforms support frame accurate seeking but most will only seek to the nearest key-frame in the video. In general to improve seek accuracy and responsiveness you can re-encode your videos with more frequent key-frames (smaller GOP). The downside of this is that the size of the video file will increase (more key-frames = larger file). We have two seek functions: Seek() and SeekFast(). SeekFast can be used when accuracy isn't important but whether it's faster or not depends on support from the platform and can

vary based on the video codec used.

8. **Hap Codec - Why don't my Hap encoded videos play?**

   If your platform is Windows desktop then this is usually because the DirectShow video API has not been selected. See the notes above about how to use the Hap codec.

9. **macOS - Publishing for Mac App Store and get error "Invalid Signature"?**

   We're not sure why this happens but we've had reports that simply deleting all of the .meta files inside the AVProVideo.bundle will resolve this issue.

10. **iOS - Can I playback a video from the Videos library?**

    Yes if the video is not content protected and you have it's URL. URL's for items in the video library take the following form:

    ipod-library://item/item.m4v?id=1234567890123456789

    You can get the URL of an item by using the MPMediaQuery class from the MediaPlayer framework.

11. **iOS - Can I playback a video from the Photo Library?**

    Yes. You need to use the UIImagePickerController to select the video. This will result in the video being processed into your app's temp folder where you can access it via the URL returned from the controller.

12. **Android - Why doesn't my huge video file play from StreamingAssets folder?**

    Files in the StreamingAssets folder for Android are packed into a JAR file and so before we can play them we must extract them to the apps persistent data folder. Huge files can take a long time to extract and copy and sometimes they could even cause the device to run out of storage space or memory. For really large files we recommend placing them the videos in other folders on the device and then referencing the absolute path to that file. This also has the added benefit of not having a copy huge files and wait ages when deploying builds to the Android device.

13. **Android - Why doesn't the video display when using Vuforia?**

    In our test Vuforia doesn't play well with AVPro Video when multi-threaded rendering is enabled. Simple disable this option in the Player Settings screen.

14. **Android - Why does my build fail with a message about DEX and zip classes?**

This can happen when there is a conflict in the Java code. We include a JAR called zip_file.jar which contain classes for handling zip files. If you have another plugin in your project that also contains these classes then Unity will fail the build. Read the exact Unity error message and if it is related to zip classes then you can try deleting zip_file.jar file.

Or you may need to upgrade from JDK 1.7.0 to 1.8.0

15. **Cardboard/Google VR - Why is the image shaking?**

    We've had reports that this is caused by enabling the option "track position" in the Gvr head.

16. **iOS - Why do I get this message about privacy permission using camera?**

    Some users have reported getting this message:
    "This app attempts to access privacy-sensitive data without a usage description. The app's Info.plist must contain an NSCameraUsageDescription key with a string value explaining to the user how the app uses this data."

    This seems to be a bug in Unity 5.4.1 and has been resolved in Unity 5.4.2.

17. **Will you support MPEG-Dash or another adaptive streaming system?**

    MPEG-Dash is currently supported on Windows 10 and Windows UWP 10 only. We hope to improve support for these sorts of streaming systems eventually. It's currently not high on our list of priorities. The plugin already supports some basic streaming protocols and for now we're focusing on basic features before we look at advanced features such as these. HLS adaptive streaming is supported on most platforms.

18. **Why is my video playback freezing on Android?**

    We have found that H.264 videos with GOP = 1 (all I-Frames / Key-frames) freeze on Android. GOP interval of 2 seems to be fine.

19. **Why do the video colours display incorrectly when using Windows hardware GPU decoding?**

    This is usually caused by having Unity set to Linear rather than Gamma colour-space. Our software decoder does handle Linear colour space but the hardware decoder doesn't. There are two ways to resolve this. Firstly you can of course which Unity back to Gamma colour-space (via Player Settings). If you require Linear colour-space then you just need to add the line col.rgb = pow(col.rgb, 2.2) to the end of any shader you use to display the video. In the InsideSphere shader we

have already added this line and it can be enabled by selecting the material in the inspector and ticking “Apply Gamma”.

20. **Why can’t I stream HLS at 4K in Windows 10?**

    Windows 10 seems to limit the stream resolution based on your screen resolution and the monitor DPI scaling setting. To get 4K using HLS you need a 4K monitor and your DPI scaling set to 100%.

21. **Why does the video control react correctly in Windows 7, but not in Windows 8 or above?**

    If you try to call video control function (eg Seek, Mute, SetVolume etc) just after you open the video file, this can cause problems. Video loading can be synchronous or asychronous, so you need to wait until the video has completed loading before issuing any control commands. In Windows 7 and below the DirectShow API is used by default. Currently the open function works synchronously, so control functions will work immediately after loading. In Windows 8 and above the Media Foundation API is used by default, and this is asynchronous. The best approach is to use the event system to wait for the video to be ready (contain meta data), or to poll the MediaPlayer (Control.HasMetaData()) before issuing control functions.

22. **What’s the difference between your different Unity video playback plugins?**

    We currently have 3 video playback plugins for Unity:
    a. AVPro Video
    b. AVPro Windows Media (deprecated August 2016)
    c. AVPro QuickTime (deprecated May 2016)

    Here is a table giving a rough comparison of the plugin features:

	AVPro Video	AVPro Windows Media	AVPro QuickTime
First Released	**2016**	2011	2011
Windows XP-7	**Yes****	Yes	Yes (with QT installed)
Windows 8-10	**Yes**	Yes	Yes (with QT installed)
Windows UWP	**Yes**	No	
Windows Phone	**Yes**	No	
OS X	**Yes**	No	Yes
Android	**Yes**	No	No

iOS	**Yes**	No	No
tvOS	**Yes**	No	No
WebGL	**Yes**	No	
64-bit	**Yes**	Yes	No
2K H.264	**Yes**	Only with 3rd party codec	Yes
4K H.264	**Yes**	Only with 3rd party codec	Yes but very slow
Streaming URL	**Yes**	Not really	Yes a bit
Hap Codec	**Yes**	Yes	Yes
MP4 Container	**Yes**	Only with 3rd party codec	Yes
Works with VR	**Yes**	Yes best with 3rd party codecs	Not great
ASync Loads	**Yes**	No	Yes
Transparency	**Yes**	Yes	Yes
Speed	**Fast**	Medium, fast with 3rd party codecs	Medium

** Currently only using DirectShow path, adding Media Foundation path soon.

Appendix B - Release History

- **Version 1.5.6 - 8 November 2016**
  - General
    - Added “Scale and Crop” scaling mode to uGUI component (replaces old “keep aspect ratio” option)
    - Added helper function for seeking to a specific frame
    - Improved MediaPlayer event handling code
    - Fixed null exception when unloading scene from the event callback
  - Windows
    - Fixed major issue using hardware decoding which caused videos to crash or not load
    - Improved DirectShow colour reproduction by using more accurate

linear colour-space and YUV conversion functions
        - Added ability to get the presentation timestamp of the current video frame - useful for more accurate frame syncing: TextureProducer.GetTextureTimeStamp()
        - Fixed issue with HLS streaming crashing when changing stream quality and resizing texture
        - Fixed AudioOutput issue where playing back multi-channel videos would cause the video not to load
        - Fixed AudioOutput issue where some videos would play back at the wrong pitch

- **Version 1.5.5 - 1 November 2016**
    - Windows
        - Fixed issue on old Windows versions without D3D11 which caused the plugin to not load
    - WebGL
        - Fixed width and height being zero when MetaDataReady event fires

- **Version 1.5.4b - 31 October 2016**
    - Fixed broken iOS, tvOS and macOS release from v1.5.4
    - Fixed “null id” error in in multi-video demo in WebGL builds

- **Version 1.5.4 - 28 October 2016**
    - General
        - 02_Demo_imGui demo scene updated to show network streaming progress
        - Improved how destroyed MediaPlayer components are shut down
        - Fixed alpha packing in background transparent shader
        - Fixed some bugs in three of the sample scene scripts that would cause a crash due to null values or invalid textures left behind when destroying MediaPlayer
        - Added documentation about hardware GPU decoding
    - Android
        - Fixed bug where IsPlaying would return false during playback
        - Fixed a shader compiler error for newer Android phones with Snapdragon chipset when using the OES shaders
        - Optimised code to generate less garbage
        - Some internal JNI caching for speed
        - Added new unlit OES shader
    - macOS
        - Added support for Linear colour space
        - Improved feedback when progressive and streamed movies are buffering
        - Added support for macOS 10.12 streaming auto wait behaviour
    - iOS and tvOS
        - Fixed a case where some HLS streams wouldn’t play

        - Improved feedback when progressive and streamed movies are buffering
        - Added support for iOS/tvOS 10 streaming auto wait behaviour
    - Windows
        - Added new AudioOutput component so that audio can be played directly through a Unity AudioSource node. This allows Unity audio effects to be used, spatial 3D movement with falloff and doppler, and most importantly allows sound to move with the head in VR!
        - Fixed a visual glitch when using the hardware decoder path, most noticeable at the end of videos
        - Streaming can now return multiple buffered ranges using method GetBufferedTimeRange()
        - Windows watermark improved for GPU decoding path
        - Potential fix for DirectShow seeking issue causing hanging on some systems
    - VR
        - Support for 3D audio via the new AudioOutput component (Windows only, supports Oculus Rift and HTC Vive)
        - InsideSphere transparency shader now supports transparency packing

- **Version 1.5.2 - 11 October 2016**
    - General
        - uGUI and IMGUI components now automatically displays videos with stereo or alpha packing correctly without having to assign material manually
        - Fixed shader build errors in Unity 5.4.1
        - Fixed shader build errors in Unity 4.x for D3D9 and OpenGL
        - Added stereo support to more shaders including uGUI
    - Workflow
        - Improved component menu layout by grouping components
        - Logging improved so some log messages can be clicked on to highlight the logging MediaPlayer
    - Windows
        - Linear colour space now automatically rendered correctly when using GPU decoder
        - Fixed memory leak in GPU decoder introduced in v1.5.0
        - Fixed old memory leak in Notify system
        - Fixed HLS adaptive resolution changing when using GPU decoder and improved texture switch to remove 1 frame glitch
    - WebGL
        - Cubemap script fixed to not show texture seam
        - Texture filtering and wrapping modes applied correctly now
        - Added crossOrigin=”anonymous” to video element

- **Version 1.5.1 - 30 September 2016**

- Windows
  - Fixed crash bug when playing audio files in DirectShow path
  - Fixed visual flicker bug in Unity editor in DirectShow path

- **Version 1.5.0 - 29 September 2016**
  - General
    - Added new shader for uGUI to support stereo videos
    - Added global option to disable logging
    - Added audio track selection to IMGUI demo
    - Fixed editor bug where "\" character could be used in file path
    - Updated documentation
  - Android
    - Added display of initial "poster" frame for videos set to not auto-play
    - Fixed bug that prevented files with escape characters from loading
    - Fixed bug causing audio files to not play
    - Fixed issue where GetCurrentTimeMs could report values greater than Duration
  - macOS
    - Fixed no error being returned for loading videos without correct plist permissions set
  - Windows
    - Hardware decoding enabled by default
    - Hardware (GPU) decoding no longer require command-line parameter
    - Better video API selection in editor
    - Added support for multiple audio tracks to DirectShow playback path via IAMStreamSelect.
    - Added Media Foundation multiple audio track code back in
    - Fixed some minor resources leaks
  - Windows Phone / UWP
    - Added option for hardware decoding (enabled by default)
  - WebGL
    - Added support for multiple audio tracks
    - Fixed frame rate and count not displaying
    - Fixed issue where multiple videos wouldn't unload cleanly
    - Cleaned up plugin code, removing unneeded code

- **Version 1.4.8 - 12 September 2016**
  - General
    - Less garbage generation
      - Optimised ApplyToMesh component
      - Optimised rendering coroutine of MediaPlayer component
      - Added global option to disable debug GUI
  - Workflow
    - Improved file path display and editing
    - Added watermark description to trial version
  - VR

        - Cubemap script (for 3:2 cubemap VR)
            - Fixed vertical flip issue
            - Improved performance
        - Added fog and transparency options to VR sphere shader
        - Android OES optimisation (see below)
        - Shaders support GLSL better
    - Android
        - New faster rendering path, less memory, faster, no overheating - see OES playback path notes above
        - Added ability to load file from within another file by specifying an offset
            - Which lets you hide video files from prying eyes
    - macOS
        - Fixed occasional memory leak when destroying player
        - Fixed editor warning about absolute path
    - iOS / tvOS
        - Fixed occasional memory leak when destroying player
    - Windows
        - HLS streams now dynamically adjust texture size
            - Resulting in support for higher quality streams
            - Fixes issue where videos would become letterboxed
    - WebGL
        - Fixed broken build issue when using the “use pre-built” build option

- **Version 1.4.4 - 20 August 2016**
    - General
        - Packed transparency support added
            - Use top-bottom or left-right packed layout to encode alpha channel
            - Set the packing type in the “Media Properties panel”
            - Supported by all display components automatically
            - See “Transparency Notes” section of this document for encoding tips
        - “Platform Specific” section now highlights platforms with modified settings
        - Minor optimisations to the ApplyToMaterial and ApplyToMesh components (used property ints instead of strings)
        - Minor optimisation to DisplayIMGUI (disables layout pass)
        - Fixed CubemapCube.cs to handle vertically flipped video textures
        - Changed many components to use UpdateLate() instead of Update() to fix some crash issues
        - Various minor improvements
    - macOS
        - Fixed compatibility issue with OS X 10.10 (Yosemite)
    - Windows
        - Huge performance improvements when using DirectShow playback path due to usage of NV12 format (D3D11 only)

        - Added support for Hap Q format (D3D11 only)
        - Added performance warning message when using software decoder instead of GPU decoder for large videos
        - Fixed bug in DirectShow audio mute not restoring volume
    - WebGL
        - Fixed issue where IMGUI text became flipped
    - Upgrade Notes
        - Some shader files have moved (and been renamed) from /Materials and /Demos/Materials to /Resources/Shaders, so make sure to delete duplicates that results from merging

- **Version 1.4.0 - 10 August 2016**
    - WebGL
        - WebGL platform support added
    - macOS
        - Now with Metal rendering support
        - The Hap codec is now supported on macOS
    - iOS / tvOS
        - Metal rendering path now supports multi-threaded rendering
        - Various fixes to rendering and memory resources
    - Android
        - Ability to adjust the playback rate (Android 6.0 and above only)
    - Windows
        - Fixed audio device override not working in builds
        - Fixed fast seeking function
    - General
        - Added support for video files containing multiple audio tracks
        - Video frame rate is now available via Info.GetVideoFrameRate()
        - Fixed some issues with events firing at the wrong time
        - Streaming videos can now query IsBuffering() and GetBufferingProgress()
        - Improved errors reporting
        - Renamed Info.GetVideoPlaybackRate() to GetVideoDisplayRate() to avoid confusion
        - New “scriptlets” have been added to the /Demos/Scripts folder which give mini examples of how to do simple scripting
    - VR
        - Fixed bug in stereo section of VR sphere shader which caused eyes to flip on some platforms
        - Fixed a bug in Windows where the audio override wasn’t being passed through for VR headphone device name
    - Workflow
        - Added new “SA” button for shortcut to StreamingAssets folder
        - Improved editor inspector performance
    - Upgrade notes
        - Demos scenes have been renamed so you should delete the old .unity

files in the /Demos folder otherwise you will have duplicate files.
        - The sample video files in StreamingAssets have been moved into a subfolder within StreamingAssets/AVProVideoDemos/ so make sure to delete the old ones.

- **Version 1.3.9 - 15 July 2016**
    - Android
        - Fixed bug for Unity versions < 5.2.0 that caused videos not to display
        - Fixed bug for Android versions < 6 that caused video pausing and other operations to break playback
        - Removed zip classes from plugin package to fix conflicts with other plugins using the same zip classes
    - General
        - Better error handling via new GetLastError() function and a new Error event type
        - NGUI support added via new component
        - TimeScale support feature now disabled by default and exposed in new Global Settings panel
        - ApplyToMesh/Material script improved with new scale and offset parameters
        - Added platform overrides for Windows Phone and Windows UWP
        - Improved documentation
    - Workflow
        - Added new Android warning for loading absolute path without external permission set in PlayerSettings
    - VR
        - Fixed VR audio output device not working
        - New high quality option for sphere shader to improve quality at the poles and general optimisations
        - Fixed issue with left/right eyes becoming swapped sometimes
        - Minor fixes to cube map script

- **Version 1.3.8 - 30 June 2016**
    - Bug fixes
        - Fixed stereo VR script broken in v1.3.6
        - Fixed issues with UWP and Hololens support
        - Fixed Windows DirectShow issue preventing Hap videos playing when another source filter is installed
        - Fixed Windows DirectShow bug in watermark for Hap videos
    - Workflow
        - Improved recent file list to be sorted by when the files were last used

- **Version 1.3.6 - 27 June 2016**
    - General
        - Added (BETA) support for Universal Windows Platform 10 and Windows Phone 8.1

        - Added (BETA) support for Time.timeScale and Time.captureFramerate allowing video playback to work properly with video capture systems that alter Unity time
        - Added ExtractFrame function to MediaPlayer
        - Added Extract Frames demo scene
        - Added SeekFast function which will try to seek to keyframes only so it's less accurate but faster. On unsupported platforms it will just do a normal seek operation
        - Added functions to change the playback rate. Note that the behaviour of this varies between platforms and is not available yet on Android
        - General API improvements, platform overrides system improved
        - Fixed bug where disabled gameObject would cause rendering to stop
        - Fixed bug where destroying MediaPlayer while uGUI component is using it could cause a crash
        - Fixed rare bug where uGUI component would become offset or hidden
        - Fixed rare bug where m_events would be null
    - VR
        - Fixed VR sphere shader coordinates on some platforms, especially cardboard
        - Added "Media Properties" section to MediaPlayer where you can define the stereo packing of a video. This then automatically adjust the sphere shader to render the video correctly per eye
        - Fixed bug in InsideSphere shader where stereo rendering would flip the eyes at some angles
        - Added support for Unity 5.4 "Single-Pass Stereo Rendering" to 360 InsideSphere shader
        - Added new VR sample with "cubemap 3x2" (facebook layout) support
    - Workflow
        - Added button in preview panel to save frame to PNG
        - Added persistent option to preserve the media player between level loads
        - Added multi-object MediaPlayer component editing support
        - Texture properties (wrap mode, filtering) can be set via Media Properties panel
        - Debug GUI now shows preview of video
    - Android
        - Fixed bug where no visuals would appear when old devices
        - Switched from AAR to JAR/SO files to improve Unity 4 compatibility
        - Added x86 support
        - Fixed bug where Pausing a stopped video would lock it up
    - OS X / iOS / tvOS
        - Large performance boost on iOS and tvOS (removed texture copy)
        - Streaming video buffering stalls now recover and IsBuffering function added
        - Seek accuracy improved
        - Looping handled better to reduce seam time

        - Fixed texture memory leak in Metal rendering path
        - Fixed mute not working with autoStart
    - Windows
        - DirectShow video playback smoother
        - Fixed DirectShow support for showing images or videos with zero duration
        - Added platform-override option to force DirectShow player
        - Added platform-override option to force audio output device. This is useful for VR hardware that has its own audio device.
        - Fixed poster frame not displaying
        - Fixed videos not displaying due to empty RGB32 alpha channel
        - Fixed D3D9 lost device issue
        - Fixed D3D9 issue of texture not being released
        - Fixed DirectShow player time displaying NaN sometimes at the end of video
        - Fixed crash bug if you closed Unity with video was playing
        - Fixed Windows N edition crash

- **Version 1.3.0 - 20 April 2016**
    - Android
        - Added multi-threaded rendering support
        - Fixed various rare bugs
    - Windows
        - Exposed GPU video player path (when using command-line argument “-force-d3d11-no-singlethreaded”), this requires Win8.1 minimum and D3D11
        - Windows XP, Windows Vista and Windows 7 support added by adding DirectShow playback path
        - Hap1 and Hap5 codec support added for Windows 7 and below (via DirectShow playback path)
        - Fixed audio only media not being seekable
    - iOS
        - Fixed iOS 8.2 compatibility issue
    - Workflow
        - Editor pause button now pauses/resumes playing videos
        - Added new ‘about’ section with handy links
        - Improvements to editor UI
        - UI fixes for standard Unity editor ‘light’ skin
    - Bugs
        - Fixed IsFinishedPlaying event not firing on some platforms
        - Fixed player freezing when component is disabled. It now pauses OnDisable() and resumes OnEnable().
        - Fixed crash in VCR demo related to uGUI component when changing videos
        - Fixed bug closing application with many videos open
        - Fixed seeking bug for audio-only media

    - Documentation
        - Updated FAQ
        - Added stereo virtual reality notes

- **Version 1.2.0 - 1 April 2016**
    - General
        - Improved support for audio-only media
        - Improved streaming support
        - Added HasVideo, HasAudio and HasMetaData functions
        - Added MetaDataReady event
        - First frame now displays without starting playback
        - Added new ApplyToMesh component
        - Removed troubled #AVPRO defines
    - Android
        - Fixed issue where Finished event wouldn't fire
        - Minimum API level bumped from 15 to 16
    - Mac, iOS, tvOS
        - improved performance of multiple video playback
    - Windows
        - Improved performance (duplicate frame handling)
        - Added missing IsSeeking function
        - Fixed IsFinished function always returning false
        - Fixed URL parsing
        - Fixed OpenGL texture cleanup
        - Fixed minor D3D11 leak caused by views not being released
        - Improved init/deinit
    - Workflow
        - Resulting file name is now displayed clearly and is copyable
        - More file extensions in inspector file browser
        - Components now have AVPro icon
        - Added warnings for incorrect Player Settings for streaming videos
        - Editor platform override tab is restored
    - Debugging
        - Improved logging
    - VR
        - Improved 360 Sphere demo support for GearVR
        - InsideSphere shader has color tint option for stereo debugging
    - Docs
        - Added video streaming notes
        - Added VR notes
        - Improved notes on supported formats

- **Version 1.1.0 - 11 March 2016**
    - Added Windows support
    - Added lat-long stereo top-bottom and left-right video support for VR
    - Added per-platform file path overrides
    - Absolute and relative path loading support added

    - Android loading from OBB support added
    - Workflow improvements
        - Added browse button
        - Added recent file list
        - Added relative folder support
        - Improved MediaPlayer UI, including preview in editor
    - Created a scripting class reference
    - Improved documentation
    - Fixed Linear colour space support in Windows
    - Fixed shutdown and memory freeing
    - Various bugs fixed

- **Version 1.0.0 - 25 February 2016**
    - Initial release on the asset store
    - Added new events

- **Version 0.9.0 - 17 February 2016**
    - Initial beta release

Appendix C - Roadmap

- **Version 1.6.0**
    - Add global audio volume
    - Better streaming support for Windows + streaming demo
    - Audio balance
    - Audio streaming back to Unity
    - Add use cases to documentation
    - Glitch free multi video player with optional transitions

- **Version X.X.X**
    - Split out media definitions from player
    - Playmaker support
    - Loading from memory/resources
    - Fallback to using MovieTexture on some platforms?
    - 10-bit HEVC support?

- **Version X**
    - ← Your suggestions here, let us know :)