

Movie Player plugin v0.11 for Unity

Copyright © 2014 SHUU Games
toomas@shuugames.com

Plays back AVI files containing MJPEG and MPNG video (not MPEG!) with PCM audio streams in Unity4 (free or Pro) and Unity5. All development and target platforms are supported, but it's meant primarily for standalone and web builds. It's probably the only serious plugin that can play movies in Free Unity.

Features

- Unity 5 and Unity 4 compatible
- Instant seek, fast load, can do runtime loading, handles huge files, works in Editor, full source
- AVI playback using a MoviePlayer component or access AVI audio and video from a script
- Simple "Movie Encoder" tool for converting movie clips into supported format (depends on ffmpeg)
- Simple "Duplicate Frame Remover" tool for further reducing AVI file size without sacrificing quality
- MovieStreamer component, that can play HTTP MJPEG streams. **Requires Unity4 PRO or Unity5**
- Implemented in managed C#, full source included
- Playmaker support

More technical details and limitations

- Up to 720p 30fps is the optimum quality on modern computers
- No file size restrictions, multi-gigabyte AVIs load and play just fine
- 1 audio and 1 video stream per AVI
- Movies can be loaded at runtime
- Longer movies can be decoded on fly, shorter clips can be preloaded into a spritesheet animation
- Fully self contained, doesn't depend on MovieTexture or other libraries
- Supports only MJPEG video, MPNG video with alpha and raw RGB(A) frames (32, 24 and 16bit)
- Supports only PCM audio (N channels, any frequency, alaw, ulaw, 16bit signed LE, 8bit unsigned)
- Plays HTTP MJPEG streams that have Content-Type: multipart/x-mixed-replace. This protocol is for low framerate video streams without an audio. "Internet cameras" often support it. Also applications like VideoLAN can output this type of stream.
- It is NOT a general purpose media player!

Getting started

1. Create a new game object and attach MoviePlayer component to it

(if the game object has a renderer and material then the video can be played back using that renderer. Otherwise screen space / full screen playback is possible)

2. Drag a movie from "Sample movies/" to Source property in MoviePlayer

(if you want to use your own movie, you may need to encode it first, unless it's already in MJPEG PCM AVI format. You can use "Window/Movie Player Tools/Movie Encoder" for this task, or run ffmpeg directly from command line:

```
ffmpeg -i in.mp4 -qscale 4 -vcodec mjpeg -acodec pcm_s16le out.avi
```

When copying it to Assets folder, add .bytes extension to clip4unity.avi so that it gets imported as a binary file)

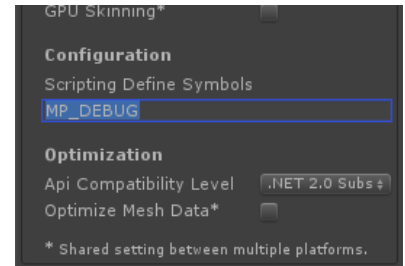
3. Check Draw To Screen and Play. Now when you press Play in Unity Editor, the movie is loaded and played back automatically.

4. While the movie is playing, you can safely adjust all MoviePlayer properties, try dragging currentTime to seek. The properties marked with (*) require reloading the movie if it's playing.

Profiling and debugging

Defining **MP_DEBUG** will enable logging. Without it only errors are logged into Console. One time action performance info, like how long it took to recreate frame index for raw MJPEG stream, can be found in there too.

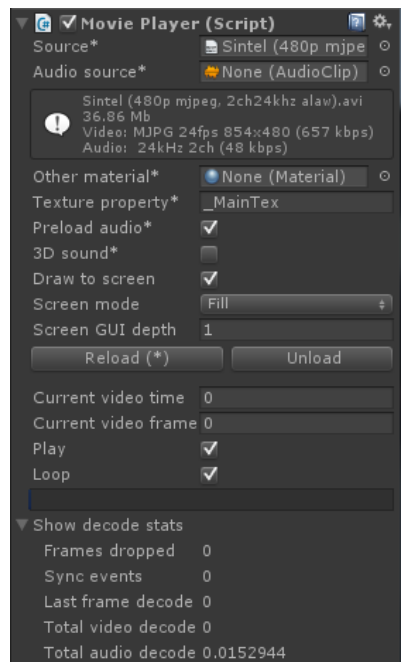
The MoviePlayer component has a foldout **Show decode stats** that displays time video and audio decoder use. Also it shows how many frames in video were skipped/dropped and how many time audio and video had to be resynced.



Most of the stats are also accessible through scripting. FeatureDemoUI.cs script in FeatureDemo scene does it.

MoviePlayer component

Source*	TextAsset (with .bytes extension) is actually the AVI container file or raw video stream.
Audio source*	If assigned, this AudioClip will override any audio in the AVI container.
Material*	If assigned, this material gets the video decoder framebuffer texture assigned to.
Texture property*	Material property name. Used for material and renderer.sharedMaterial
Preload video*	If TRUE, video frames are decoded into spritesheet when the file is loaded. Use it only for short videos.
Preload audio*	If TRUE, all the audio is decoded into memory when the file is loaded. Use it only for short videos.
3D sound*	If TRUE, the audio playback buffer will be created as 3D sound, otherwise it's 2D stereo.
Draw to screen	If TRUE, the video frames will be drawn to screen using DrawTexture in OnGUI
Screen mode	Screen space use
Custom rect	If screen mode is CustomRect, then this is used
Screen GUI depth	Change this to avoid wrong draw order.
Current video time	Use for seeking, works when the movie is either playing or stopped.
Current video frame	Use for seeking, works when the movie is either playing or stopped.
Play	Use for it for starting and stopping the movie.
Loop	If TRUE, the movie will rewind and start from start when finished playing.
(re)load button	First use case - if you change the source or any properties marked with (*) when in play mode, you need to load it again for those changes to have an effect. Playback won't stop if you do this. Second use case - press it if you want to load the movie in editor, to take a look at demux, video and audio decoder info.
unload button	Releases resources related to the currently loaded movie. Most useful when in edit mode.

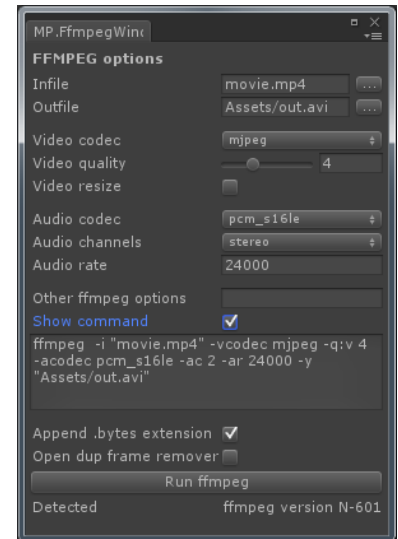


Encoding movies for playback

To use your own movies with this plugin, they need to be encoded to any supported format first. Using FFMPEG is the easiest way to do it, but other encoders work too (VirtualDub, Avidemux, etc).

For those frightened by ffmpeg command line, there is an UI which builds the command with reasonable defaults under **"Window / Movie Player Tools / Movie Encoder"**. It limits the ffmpeg arguments so that the output will be supported by this package. Tick "open duplicate frame remover" if you want to further compress the file after ffmpeg has finished.

If command line is your friend, then the simplest syntax for encoding your movie clip would be:



```
ffmpeg -i in.avi -vcodec mjpeg -acodec <supported_acodec> out.avi
```

Some useful arguments (`ffmpeg -help` for full list)

<code>-an</code>	Don't encode audio stream altogether
<code>-qscale 4</code>	Sets video quality. Smaller value = better quality and larger file. Value 4 is a good compromise. Usually instead of reducing the quality it's better to resize the video to something smaller. It helps to avoid blocking artifacts.
<code>-vf scale=854:480</code>	Resizes the video.
<code>-ac 1</code>	Include only one audio channel into the resulting file (mono)
<code>-ar 24000</code>	Resample audio to lower frequency

Supported audio codecs (`-acodec` option):

<code>pcm_alaw</code>	8bit quantized. Good for speech or music where quality is not paramount
<code>pcm_mulaw</code>	Same as <code>pcm_alaw</code> , but with another quantization table
<code>pcm_u8</code>	8bit unsigned. Very low quality, may be useful for the lo-fi feel of it
<code>pcm_s16le</code>	16 signed little endian. "Lossless audio" for all cases where the size doesn't matter

Besides all that FFMPEG can do, VirtualDubMod can also write raw formats (ARGB32, RGB24 and RGB555). Together with ffdshow codec it can also encode MJPEG video streams.

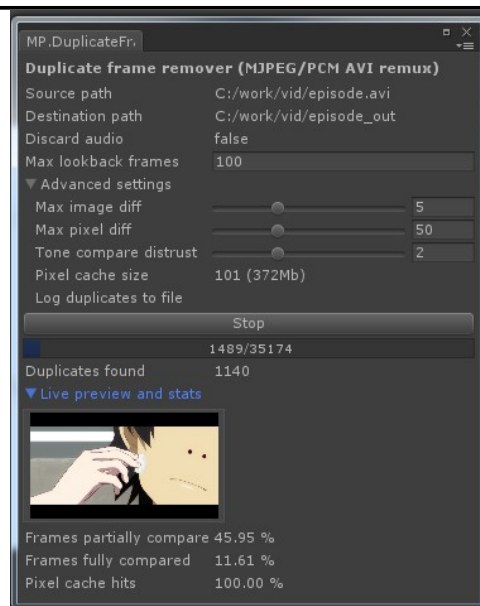
Duplicate Frame Remover

This tool reads an AVI file prepared for playback with this package and remuxes it so that all the duplicate frames are stored once (it does this by modifying the AVI index). Depending on the content, significant file size reductions can be achieved. For example, consider a stop motion movie where the AVI itself is 30fps, but frames actually change less often, but it's not possible to reduce AVI framerate, because at times there are smooth camera transitions in the movie. This movie would be a good candidate for this type of processing.

For best results, duplicate video frames don't have to follow each other, but can be up to a specified number of frames apart. They can't be arbitrarily apart, because it would cause random disk reads which is not good (a default 100 frames gives 3.3 seconds @ 30fps look back time). This makes it possible to compress anime-like clips where someone talking is animated with, say, 4 different frames where only the mouth shape changes.

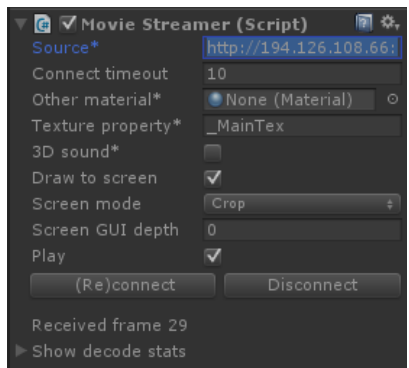
The AVI files resulting from this operation are playable by other media players like VLC, but since it's they are not "standard" any more, it can't be guaranteed.

Source path	Source MJPEG AVI file with optional audio
Destination path	Destination location for remuxed AVI
Discard audio	When checked, there will be no audio
Max lookback frames	How many frames back from current playhead position we search duplicate frames from
Max image diff	Sets the upper limit how much all the pixels on compared frames can differ in average, but still be considered the same. If it's too low, duplicates are not detected. If it's too high, some similar frames that are not duplicates will be considered erroneously as duplicates
Max pixel diff	Sets the limit on how large a pixel value difference can be so that the frames are still considered the same. This is required, because sometimes the frames are very similar and pass "max image diff", but have significant pixel changes. For example in an animated movie a main drawn character is further away from a camera, but is talking with his tiny mount. If this value is too high, different frames may be erroneously considered the same. If it's too low, duplicates are not detected
Tone compare distrust	For speeding up the process, average tone of the two frames is compared. These are just two colors. This value here determines how much we should trust this comparison. Lowering this value increases processing speed but reduces accuracy (yes, more trust is lower value!). Increasing this will slow down processing, because more frames are compared pixel by pixel possibly giving better results
Pixel cache size	Sets how many decoded video frames are cached by keeping all the pixels in memory. Bigger cache increases speed, but can take ridiculous amounts of memory. Usually a good balanced value here is 1x-2x "max lookback frames"
Log duplicates to file	An optional text file into which data about duplicate frames is written. Use it for debugging and your own purposes
Process file / Stop	Starts or stops the processing. The movie is processed sequentially and you can stop it any time without corrupting the output file. During processing some properties can be changed. For example if you see slowdown and "pixel cache hits" percent is declining then you may want to temporarily decrease "max lookback frames" or lower "tone compare distrust" value (that's more trust)



MovieStreamer component (requires Unity4 PRO or Unity5)

This component can connect to network streams and play back content. Currently this is very limited, supporting only HTTP MJPEG streams sent using Content-Type: multipart/x-mixed-replace. There's no audio and framerate is inconsistent and low due to the protocol. When using this component (or underlying Streamer), please keep in mind that it is a subject to access policies imposed by Security Sandbox, especially in Web Player builds (<https://docs.unity3d.com/Documentation/Manual/SecuritySandbox.html>)

Source*	Stream URL	
Connect timeout	Connect to URL timeout in seconds	
Material*, Texture property*, 3D sound, Draw to screen, Screen mode, Custom rect and Screen GUI depth work exactly like in MoviePlayer component		
Play	If TRUE, then the framebuffer texture will be updated once the next frame is received. Setting it to FALSE doesn't disconnect from a stream. You need to explicitly call Close from a script to close the connection (or press Disconnect button)	
(Re)connect	Connects to the URL. Use it to reconnect in case the connection is lost	
Disconnect	Disconnects from a stream and releases resources	
<status>	Status line lets you see what the background thread is doing. Normally you will see there messages about what frame was received, but it'll also show exceptions which stop the background thread	

Technical notes for specific build targets or Unity versions:

- Unity 4.3 and OSX build targets. WebCamTexture is not available so local webcam streaming is disabled. The support for WebCamTexture is expected to come back some day.
- Web player build. Since it's not possible to use HttpWebRequest a lower level TcpClient is used to make HTTP connections. Because of that only "200 OK" responses are accepted and all response headers are ignored.
- Web player build. The actual connection timeout may take longer than specified, because behind the scenes Unity will make another connection to fetch security policy.

Playmaker support

Playmaker actions cover most common use cases like playing back a video, seeking, sending event when a video clip has finished playing, reading video metadata, etc. It makes loading a video from various sources easy (filesystem, Resources folder, StreamingAssets folder, binary text asset or using asynchronous WWW download).

To install Playmaker actions and a sample scene, import **PlaymakerActions.unityEngine**

Transparent movies

For making videos transparent there are two choices: alpha channel or chroma key. MJPEG format doesn't support alpha channel, therefore chroma key also known as greenscreen technique needs to be used. For this there is "Unlit / Transparent ChromaKey" shader included in this package.

True alpha channel transparency can be achieved by using MPNG video codec or an uncompressed format. Downside of this is that MPNG is much harder to decode which limits usable video resolution. Uncompressed formats usually mean unpractically large files.

Scripting overview

There are 2 components: MoviePlayer and MovieStreamer. For normal use, probably all you want to do is to control properties of those two components from your scripts. If you find that this too limiting, then you can write your own playback components based on MoviePlayerBase.

Those components use abstract class Demux, VideoDecoder and AudioDecoder implementations. A demux is responsible for extracting elementary streams from a container format like AVI. To a video decoder you tell that I want the next video frame (or some specific frame) to be put into a framebuffer Texture2D. Then the decoder uses a demux to read encoded bytes for that specific video frame and decodes those into framebuffer pixels. An audio decoder does the same thing, but works with samples (*float[]*) directly compatible with AudioClip.

Besides playback, some of the less used functionality can only be accessed from scripts. One of those is sequential remuxing. A Remux is again, an abstract class. Currently there is only one remux for AVI. After you've initialized a remux and provided video stream info, you can call methods that add encoded video frames and audio samples to it. At some point later, when you close the remux, you'll have a playable AVI file. Currently there are no encoders in this package, so you have to provide encoded bytes that are read directly from a demux or use unencoded RGB format.

This package provides Streamer class which can play back "streams" that are not seekable. A Streamer is an abstract class inheriting from Demux. A Streamer is normally used to connect to a network stream, but it could also be used to for local content generation like local webcam capture or generating procedural videos.

DuplicateFrameFinder is another rather optimized class worth mentioning that you may want to use from your code. It is used internally by "Duplicate Frame Finder" tool. Compared to naïve frame comparison it has many improvements. It provides fuzzy matching, the pixels doesn't have to be exactly the same, but instead you set overall average and max pixel difference. For speeding up, it won't compare all the pixels all the time. Instead it uses tone comparison to get the first idea about similarity, then it uses only a portion of pixels scattered around the image for comparison and only if the images are similar enough, all pixels are compared.

In this package there is a **ScriptingDemo** scene using **ScriptingDemoUI.cs** script which demonstrates accessing the API in various ways.