


# OSX VIDEO TEXTURE PRO

QUICKTIME  
FOR UNITY3D



Developed by Brian Chasalow

<http://www.chasalow.com>

OSX Video Texture Pro is a HD video player plugin for Unity Pro, using fast path GPU-based texture decoding and copying using QTKit/Quicktime on Mac OSX 10.6 or greater. Supports Unity 3.4 and 3.5. Designed for Unity developers making Mac OSX games, art installations, performances, and simulations, and for those needing a high performance, flexible video player.

## Features:

- Videos are all preloaded and paused initially, for fast, smooth playback of multiple videos simultaneously.
- Supports alpha channel transparency, and any video format that Quicktime supports.
- Adjustable volume/speed/timeline scrubbing.
- Efficient playback - only renders new frames when they are available from a video.
- Callbacks generated in Unity when a video ends.
- Playlist support- queue up a set of multiple videos, and it will play them in order.
- Loop the entire playlist, or just a single video.
- GUI included that allows you to control all aspects of all video instances in your scene at runtime.
- Can render to either a RenderTexture or Texture2D.

Video walkthrough at : <http://youtu.be/DywusDnbOoc>

## Release Notes:

### Please back up your project before importing a new version!

Version 1.1.1 – Feb 1 2012

Fixed a bug in the queue list when deleting a video from the queue.

Fixed asynchronous movie loading - now if it cannot load a movie asynchronously, it will attempt one time to load it synchronously.

Added a method to support auto-resizing of full screen video playback via Aspect Fit or Aspect Fill - see Backdrop.cs. Alternatively, you can use PlaneScaler.cs as a simple method to rescale a mesh. Implement your own via the updateAspectRatio() callback.

Version 1.1.0alpha – Jan 15 2012

updated VideoTexture.AddAVideo() so that if there is no video in the queue at the time in which you add it, it will attempt to draw it to the texture.

Added .mp4 to VideoManager.validFileTypes[]

VideoManager.videoLoaded() only calls beginPlayback() if videos.Count > 0

VideoManager.checkForInvalidMoviesInQueue() is now enabled by default

Version 1.0.9 – Jan 15 2012

fixed a memory leak in the plugin. released that fix a little early in the rush ;-)

Version 1.0.8 – Jan 15 2012

**CRUCIAL FIX-** updated plugin .bundle with texture matrix stack fix. Should have better performance now and works in 3.5b6 trial. Please update to this version asap.

Version 1.0.7 – Jan 14 2012

Updated Docs with VideoTextureSimple info and VideoManager init sequence

Version 1.0.6 – Jan 13 2012

**Plugin .bundle:** - web streaming should (actually) work now.

Version 1.0.5 – Jan 12 2012

**VideoObject.cs:** VideoObject.getVideoTotalFrames(), setVideoTotalFrames(), getVideoDuration, setVideoDuration, and setVideoProperties() : exposes properties that should have been exposed.

**Plugin .bundle:** bugfix re: http web streaming in plugin bundle.

**ExampleScripts Folder:** Renamed createFromScript.cs to createFromCSharp.cs. Added createFromJavascript.js, provided as a demo for controlling videos from javascript.

**VideoManagerInspector:** fixed regression to the display of a successful video load

Version 1.0.4 – Jan 11 2012

**VideoTextureSimple.cs:** newly added, additional way to load videos, if you want to load videos from multiple locations on the hard drive. This script inherits from VideoTexture.cs, but additionally internally manages/owns its own VideoObject. It can only control the single video that it owns. Useful if you don't want to deal with

queue lists/don't want to preload any videos in your game. You may choose to preload videos in your manager for use with VideoTexture, and simultaneously use VideoTextureSimple to load other videos elsewhere on your hard drive.

**VideoManager.cs:** Start() is now Awake() to make sure it gets called before any of the VideoTexture scripts.

**VideoManager.cs:** removed currentlyLoadingIndex, added a videosToLoad list that is kept up to date with names of videos that are remaining to be loaded.

**VideoManager.cs:** changed initialization routine, removed initVideoPlayback(). Now, each VideoTexture instance will be initialized *immediately* after each video loads successfully, via beginPlayback(). It will also then begin playing if it is set up to do so. If you want to wait until all movies are done loading before playing your movies, set them to be paused initially and set them to play after videoManager.isInitialized.

**Plugin .bundle:** fixed a divide by zero error when calling setVideoFrame(0) due to modulus % 0 operation. oops!

**ExampleScripts folder:** two examples for how to use scripting to interact with/ create videos programmatically. [see createFromScript.cs and setSpeed.js](#)

**http streaming support:** feature in beta, please see createFromScript.cs for an example for how to use via scripting, using VideoTextureSimple.cs, and let me know if you have any issues. Currently supports http:// prefix- let me know if you need to use a different prefix/protocol that quicktime supports

Version 1.0.3 – Dec 21 2012

Internal refactoring + manual additions

# Table of Contents

## QUICK START GUIDE

## IMPORTANT NOTES

## THE VIDEO MANAGER OBJECT

USING THE MOVIERESOURCES.BUNDLE TO STORE YOUR MOVIES. ....	4
USING A CUSTOM FOLDER FOR YOUR MOVIES. ....	4

## THE VIDEOTEXTURE OBJECT

SETTING VIDEO OPTIONS .....	6
USING THE QUEUE LIST.....	6
VIDEO PLAYBACK.....	7
VIDEO CALLBACKS .....	7
VIDEOMANAGERSIMPLEGUI .....	8

## SCRIPTING REFERENCE

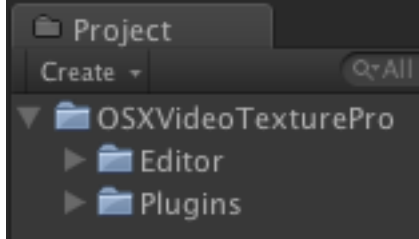
VIDEOMANAGER.CS .....	9
VIDEOMANAGER.CS NATIVE PLUGIN METHODS .....	10
VIDEOTEXTURE.CS .....	10
VIDEOTEXTURE.CS NATIVE PLUGIN METHODS .....	13
VIDEOOBJECT.CS .....	14

## COMMENTS

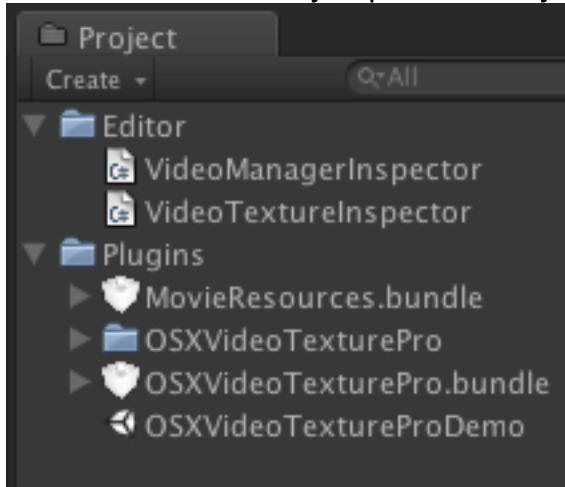
## CREDITS

## Quick Start Guide

When you download OSX Video Texture Pro, all files may be in a /OSXVideoTexturePro parent directory as shown.



You need to copy the Editor and Plugins directory to the root of your project, as shown below. It is very important that your folder layout is setup like this.



Now open **OSXVideoTextureProDemo.unity**.

Hit Play.

Look at the **VideoManager** script on your Main Camera.

Look at the **VideoTexture** script on the Plane.

alternatively,

Open **OSXCreateFromScriptsDemo.unity**.

Hit Play.

Look at the **VideoManager** script on your Main Camera.

Open the **createFromCSharp** script and look at the code- it generates a mesh plane and puts a **VideoTexture** script on the gameObject.

The second half of **createFromCSharp** script has code to generate a **VideoTextureSimple** instead of a **VideoTexture** object- uncomment that section to compare the two. Read more info in the Important Notes section about the difference between **VideoTexture** and **VideoTextureSimple**.

### To manually setup from a new project:

- 1) Add the **VideoManager** script to your MainCamera.
- 2) Add a **VideoTexture** script to your GameObjects that already have a mesh and a material. Create as many **VideoTextures** as you want – you can simultaneously play as many videos at the same time as your computer can handle.
- 3) Setup your videoDirectory in the **VideoManager**.
- 4) Add files to your **VideoTexture** queue lists.
- 5) Hit Play.
- 6) The **VideoManager** will preload all movies via openVideoPlayer()
- 7) Use the **VideoManagerSimpleGUI** script to control videos at runtime

## Important Notes

### VideoTexture vs VideoTextureSimple:

Each **VideoTexture** maintains a queue list of videos that could be played back. The videos that can be added to the queue list are preloaded by the VideoManager. The video objects preloaded by the VideoManager are shared between all VideoTexture objects in your scene.

Each **VideoTextureSimple** loads a single movie from disk - it owns this movie and does not share its movie with the VideoManager's video pool. It will unload this movie from ram when it is destroyed. It can only playback its own movie it has loaded. It also does not display a queue list. It derives from VideoTexture, and can stream http web movies- see createFromCSharp.cs for an example of this.

### File Types:

VideoManager.cs is currently setup to only see files with the extensions '.avi', '.mov' and '.m4v', but Video Texture Pro supports any filetype that Quicktime supports. To add additional file extensions, simply add additional values to the array validFileTypes in VideoManager.cs:

```
private string[] validFileTypes = new string[] { "*.avi", "*.mov", "*.m4v" };
```

Valid File Types

### Handling Aspect Ratios:

As part of the provided demo, VideoTexture.cs is configured to reset the aspect ratio of a square mesh to a video's aspect ratio by scaling its transform's x and z values. If you want to control this behavior yourself, see further instructions in VideoTexture.resetAspectRatio()

**Texture Types:**

You may either use `RenderTexture` or `Texture2D` to render videos.

`RenderTextures` are much more efficient. There is almost no reason to render to a `Texture2D`, but the option is there. The `VideoTexture` script will demonstrate how each video can be rendered to an appropriate texture type.

**VideoManager Initialization Sequence:**

(This is a little code-heavy, written for people who might want low level control.

Skip this section if it doesn't interest you)

A series of methods are triggered in the `VideoManager`'s initialization sequence. It looks like this:

- 1) `VideoManager.openVideoPlayer()`  
This begins the loading sequence.
- 2) `VideoManager.preLoadAllFiles()`  
Each video will be loaded one by one, until they are all loaded.  
The first video in the `VideoManager`'s `videosToLoad` list is loaded.
- 3) `VideoManager.videoLoaded()`  
Called back from the plugin, to describe success/failure of the video's load.
- 4) `VideoTexture.beginPlayback()`  
Called by the `VideoManager` on any `VideoTexture` object in the scene that has this video as the first video in its queue.
- 6) If there are more videos to load:  
The `VideoManager` will preload the next video, and go back to step 3.  
else, go to the next step.
- 7) `VideoManager.isInitialized = true`  
All the videos to be preloaded are loaded.

## The Video Manager Object

You are required to have this script on your main camera. It manages your video instances, and handles preloading all the videos so that playback triggering will be fast and non-blocking. First, put a VideoManager.cs script on your MainCamera.

### *Using the MovieResources.bundle to store your movies.*

You should see what is displayed in Figure 1. The button labeled **‘using MovieResources bundle. click to change’** is telling you that all movies will be loaded from the MovieResources.bundle located in your project’s Assets/Plugins directory.

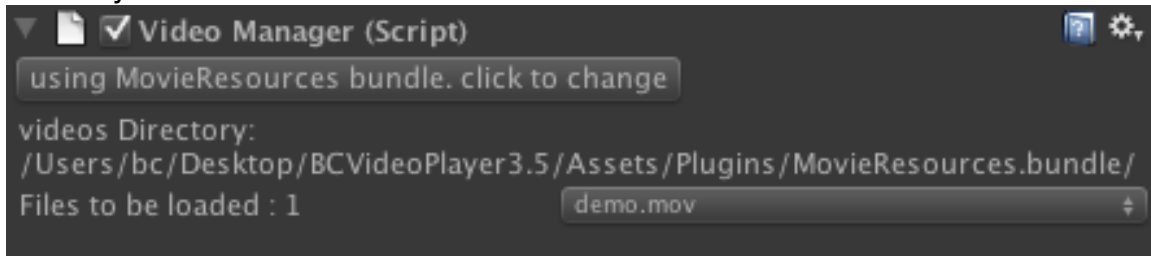


Figure 1 - The Video Manager Object

A **‘.bundle’** is just a normal folder with a **.bundle** extension, but lets you store movies in it in such a way that all your movies will get copied into the application when you build your app. Your **MovieResources.bundle** *must* be in the **/Plugins** root folder or it won’t get copied to your built app. To add additional movies to your **MovieResources.bundle**, in Finder, right click the **.bundle** and click **‘show package contents’**. You may copy any movie files into this folder.

### *Using a custom folder for your movies.*

Click the button labeled **‘using MovieResources bundle. click to change’**, and you should see a window like Figure 2. Now you should select a valid folder on your hard drive.

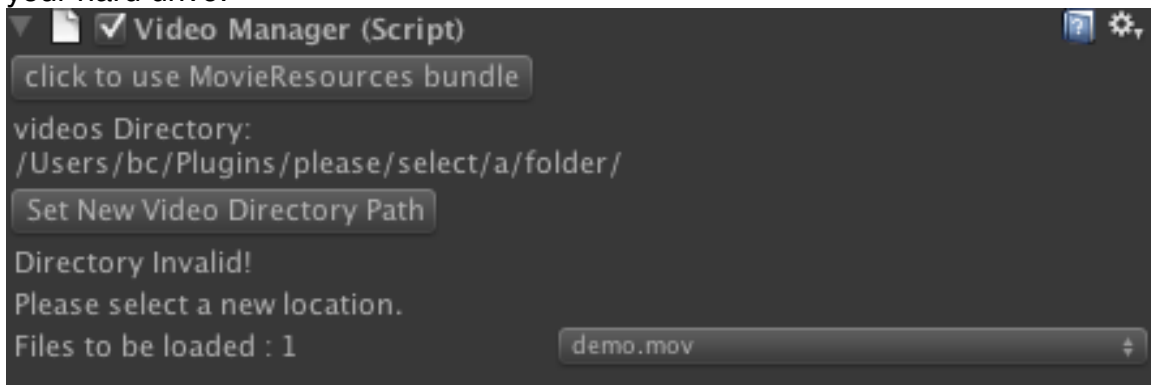


Figure 2 - Select a new folder



Click **'Set New Video Directory Path'** and point to a valid folder on your hard drive where videos exist.

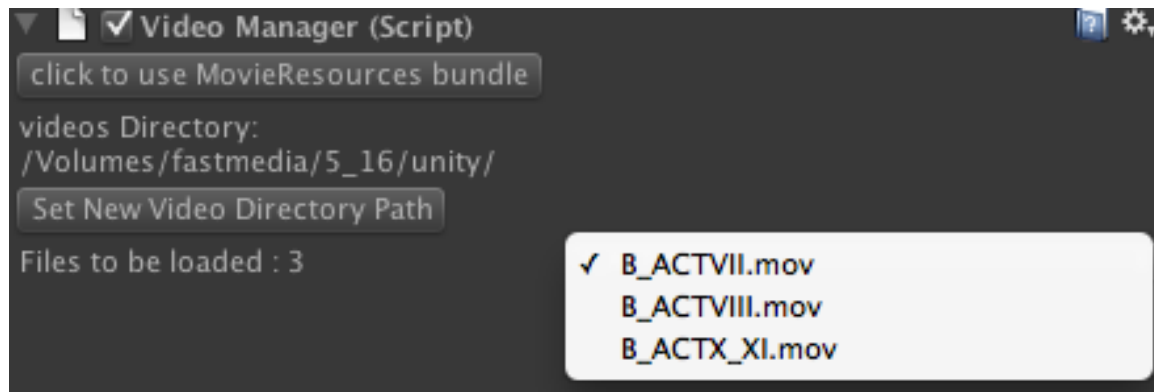


Figure 3 - valid folder is selected

## The VideoTexture Object

Create a GameObject with some sort of mesh on it. This could be a simple plane, or a complex 3d shape. Add a material to the object and select a shader. The Unlit/Transparent shader is recommended for video playback, but you may use any shader. Add a VideoTexture.cs script. You will see something like Figure 4.

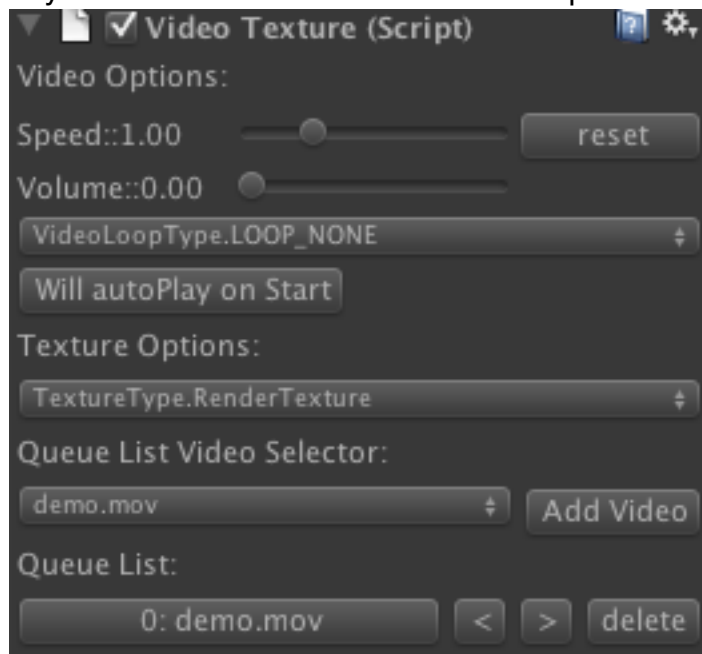


Figure 4 – The VideoTexture Object

## Setting Video Options

There are several video options, all of which you may set either before running the game, or at runtime.

**Volume:** adjustable volume control.

**Speed:** adjustable playback rate.

**Will autoPlay on Start:** pause / play controls. Can set it to be paused on start, and you can trigger playback manually later on.

### VideoLoopType:

LOOP\_NONE: video reaches the end of its playback and pauses on the last frame.

LOOP\_NORMAL: single video loops continuously. No callbacks are triggered.

LOOP\_QUEUE: video instance plays all videos in queuelist, and loops back to beginning of queue, continuously.

### Texture Options:

RenderTexture: uses less VRAM, more efficient.

Texture2D: pads to one higher texture size.

## Using the Queue List

### Queue List Video Selector:

This is a dropdown list displaying all the videos that you could add to the queue list. Your VideoTexture needs at least one video in the queue to play. Select a video from the dropdown and click 'Add Video.' Your queue list on your video instance should look something like Figure 5.

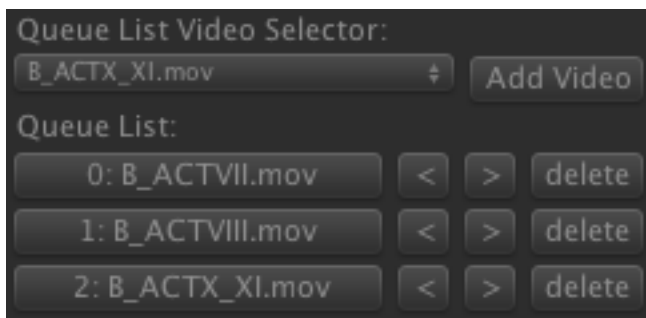


Figure 5 – the queue list

### Queue List:

**< button:** moves a video up in queue.

**> button:** moves a video down in queue.

**delete:** removes a video from the queue list.

After your folder of movies has been preloaded, and your queue list is properly setup, there are two ways to tell a video to start from your script. If you wanted to start “B\_ACTVII.mov” as shown in Figure 5, you could either call:  
`drawVideoToTexture(0);` because it is index 0 in the queue list,  
or  
`drawVideoToTexture (“B_ACTVII.mov”);`

### ***Video Playback***

When you start your game, additional features like the timeline become visible. The timeline slider allows you to scrub through a video’s timeline. The inspector also shows you the name of the currently playing video, the internal video resolution of that video, and the index number in the queue list in which that video exists.

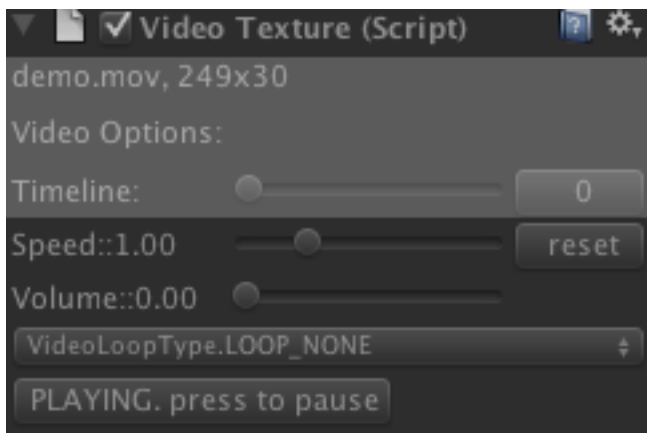


Figure 6 – Additional video options (As shown highlighted) available when Editor is Playing

### ***Video Callbacks***

Each videoTexture will call `videoEnded()` on its script instance when each video ends, as long as the instance is set to `LOOP_NONE` or `LOOP_QUEUE`. If the VideoTexture is set to `LOOP_NORMAL`, the instance will not receive `videoEnded()` callbacks. You can use the `videoEnded()` callback to cause additional things to happen in the scene every time a video loops.

When a video instance is loaded and its movie resolution does not match the current texture size of the video instance texture, the method `resizeTexturesFromVideoResolution()` is called. You can use this method to handle adjusting your mesh accordingly, to fit the aspect ratio of the movie. `resetAspectRatio()` is provided as an example of how to do this. You may comment this and handle it in a different manner.

## VideoManagerSimpleGUI

This script should be applied to your MainCamera. It is a companion tool for the VideoManager, for runtime control of any video instances in your scene. It gives you the same controls as shown in the VideoTexture inspector, and it also provides an example for how to reload your videos from disk, if you should desire to change your preloaded videos directory at runtime. By default, it is setup to hide/show the GUI if you press the 'g' button (g stands for GUI!)

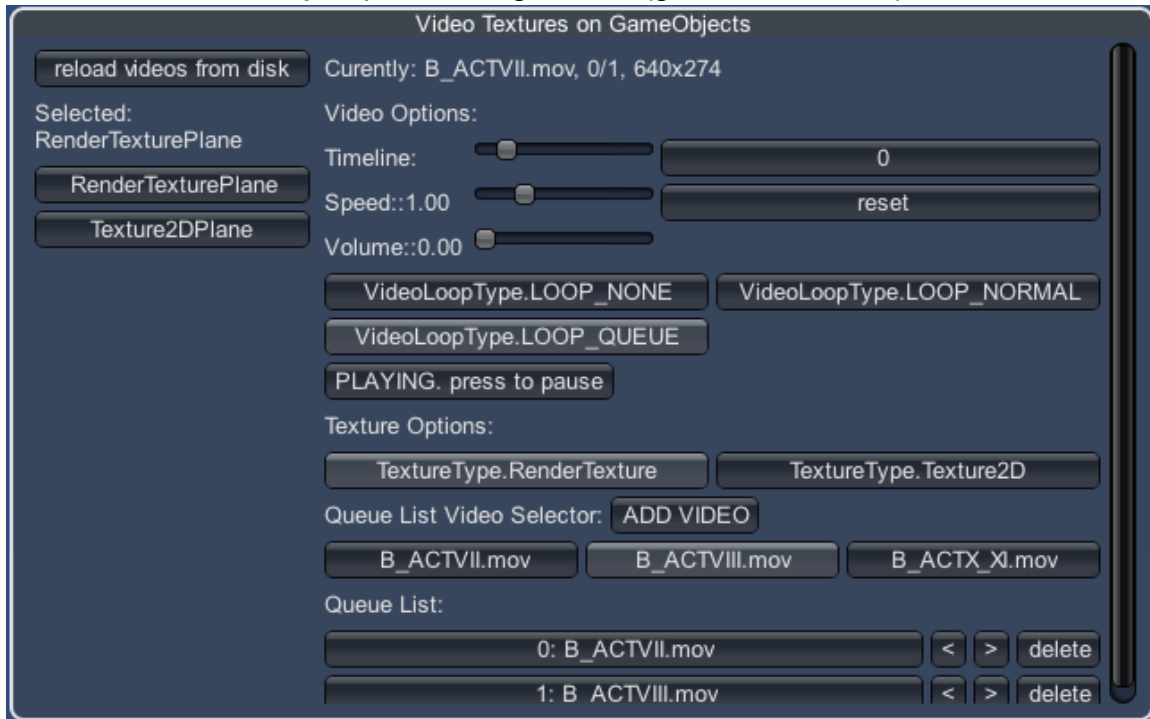


Figure 7 - VideoManagerSimpleGUI

## Scripting Reference

### ***VideoManager.cs***

You are required to have this script on your main camera. It manages your video instances, and handles preloading all the videos so that playback will be fast and non-blocking.

#### **public static void videoLoaded(IntPtr myptr);**

This will be called remotely from the plugin. 'myptr' refers to the instance of the movie if it is found. If myptr is null, the plugin is saying the movie didn't load successfully. This method then attempts to load the next movie if there is one in the queue to load.

#### **public static void videoEnded(IntPtr myptr);**

This will be called remotely from the plugin when each video playing back ends.

#### **public void addInstance(VideoTexture addMe);**

Adds a new videoTexture when a GameObject with a VideoTexture.cs script is created.

#### **public void removeInstance(VideoTexture removeMe);**

Removes a VideoTexture when one is destroyed.

#### **public void refreshVideoInstanceLinks();**

Finds all VideoInstance.cs scripts in the game and make sure they are up to date.

#### **public void checkForInvalidMoviesInQueue();**

This method compares the folderDirObjects with each VideoTexture's queuelist, and removes any objects from the queuelist that do not exist in the folderDirObjects array. This method is enabled by default but you may disable it in your VideoManager Start method if you wish, as well as VideoManagerInspector.cs's OnInspectorGUI method.

#### **void Start();**

Initializes callbacks (this is always necessary) and opens the video player

#### **public void openVideoPlayer();**

Initializes the video path, calls refreshVideoInstanceLinks, listDirectory, and begins to preload all the video files in folderDirObjects.

#### **public void closeVideoPlayer();**

Kills the video player. Call this and then openVideoPlayer again to reload.

#### **public int listDirectory(string myDirectory);**

Finds all the files in myDirectory and updates folderDirObjects with all the files.

**public void preLoadAllFiles();**

Begins to load videos in the folderDirObjects list. videoLoaded() will return from the plugin and continue with loading, movie-by-movie until are are loaded.

**public void drawVideo();**

Updates the context and video rendering for each video. Must be called from OnPreRender().

**bool initialized;**

When initialized is true, all movies to be preloaded are loaded.

**bool videosAreLoading;**

While videosAreLoading is true, the VideoManager is in the process of preloading videos.

***VideoManager.cs Native Plugin Methods***

**void killOSXVideoTexturePro();**

internal plugin shutdown mechanisms. To be called from closeVideoPlayer()

**void initVideoPathEditor(string whatever);**

Initializes the path to where all your videos are stored.

**void initVideoPath();**

Initializes a path local to your .app's MovieResources.bundle

**void initCallbacksEditor(string myPath);**

Sets up callbacks if in the editor.

**void initCallbacks();**

Sets up callbacks if in a built app.

**void updateOSXVideoTexturePro();**

if using Unity 3.4 or lower, updates the plugin and movie rendering with this single method.

**void initVideo();**

If using Unity 3.4 or lower, initializes the video plugin.

**void pluginUpdate();**

If using Unity 3.5 or higher, updates the plugin and movie rendering in conjunction with GL.IssuePluginEvent(updateContext);

***VideoTexture.cs***

This script is used to render a video texture on a material. It will generate a RenderTexture or Texture2D based on your selected script variables.

**public void updateTextureType();**

Sets up either a RenderTexture or Texture2D based on myTextureType

**public void addAVideo(string name, int i);**

Add a video to the queue list

**public void moveUp(int selectAVideoIndex);**

Move a video up in queue

**public void moveDown(int selectAVideoIndex);**

Move a video down in queue

**public void removeAVideo(string name, int i);**

Remove a video from the queue list

**void Start();**

Adds the video instance to the manager with addInstance() and initializes the video texture cache with createVideoTexture()

**public void setupTex2D();**

Sets up a Texture2D, and applies it to the material's mainTexture (optional, if you want to apply it somewhere else you may)

**public void setupRenderTex();**

Sets up a RenderTexture, and applies it to the material's mainTexture (optional, if you want to apply it somewhere else you may)

**private int getPOT(int inputVal);**

Returns the POT(Power of Two) texture size 1 up from inputVal. (I.E. inputVal is 640, returns 1024)

**public void resetAspectRatio();**

This is a demo method for how to change aspect ratio based on video width/height. It is called from resizeTexturesFromVideoResolution(), and is optional

**public void resizeTexturesFromVideoResolution();**

This is called after you change the video playing on the current object. If you want to change something based on the video's texture size, change it here.

**public void refreshTextureValues();**

This updates the cache for texture rendering- these values will be acted upon later from the GL thread.

**void beginPlayback();**

Loads video 0 in the queueList as an initial start of playback. Called from the video manager on each VideoTexture instance when it is successfully loaded. Also called on VideoTextureSimple instances

**bool isAnyoneElseUsingThisVideo();**

Makes sure no other VideoTextures are currently playing the video that you're playing

**public void drawVideoToTexture (string loadMe);****public void drawVideoToTexture (int arrayIndex);**

This attempts to load a video from the preloaded video cache using an array index value or a string name. It stops the old video if nobody else is using it, calls a method to resize the texture if necessary, starts or stops the video based on isPaused, and applies other video properties to the new video.

**public void drawGUI();**

Draws a GUI to control video instance parameters

**public void drawTimeline(bool drawTime);****public void drawCurrentlyPlayingDetails();****public void drawIsPaused();****public void drawIsPausedEditor();****public void drawSpeed();****public void drawVolume();**

VideoInstance property drawing method

**public void videoEnded();**

This method is called from the plugin when a video ends. You can use this method to perform additional game logic when a video in the queue ends.

**public void nextVideo();**

This method is called by videoEnded() to trigger the next video in queue.

**public void destroyVideoTexture();**

Only called OnDestroy()

**void OnWillRenderObject();**

Calls renderTextures() the proper time in the render chain. Videos must only be rendered from this callback.

**void renderTextures();**

Finally renders the video to the texture.



## ***VideoTexture.cs Native Plugin Methods***

**IntPtr createVideoTexture();**

Create a new video texture cache. Should only ever be generated once for each script, in Start()

**void killVideoTexture(IntPtr videoTextureInstance);**

kills the video instance (only use on quit or object-disable)

**bool bindTextureValues(int textureId, int width, int height, int scaleMode, IntPtr videoInstance, IntPtr videoTextureInstance);**

Used to pre-cache the GL rendering options for rendering to a texture. bindTextureFBO() is deprecated.

**float getSpeed(IntPtr videoInstance);**

Gets the current playback speed at which the video is set.

**float getPosition(IntPtr videoInstance);**

position of video. Position divided by duration goes from 0 to 1

**float getDuration(IntPtr videoInstance);**

duration of video.

**int getCurrentFrame(IntPtr videoInstance);**

Gets the current frame of the video.

**int getTotalNumFrames(IntPtr videoInstance);**

Gets the total number of frames in the video.

**int getLoopState(IntPtr videoInstance);**

Gets the loopState of the video.

**bool getIsMovieDone(IntPtr videoInstance);**

Gets whether the movie has ended or not. Not very useful anymore now that callbacks are integrated into the plugin.

**int getWidth(IntPtr videoInstance);**

Gets the video's internal width.

**int getHeight(IntPtr videoInstance);**

Gets the video's internal height.

**bool isLoaded(IntPtr videoInstance);**

Asks if the movie is loaded- not very useful now with callbacks.

**bool isLoading(IntPtr videoInstance);**

Asks if the movie is loading – not very useful now with callbacks.

**void setPaused(IntPtr videoInstance, bool isPaused);**

Sets whether the video should be playing or paused.

**void setVolume(IntPtr videoInstance, float myVol);**

Sets the volume from 0.0f to 1.0f.

**void setLoopState(IntPtr videoInstance, int loopState);**

Sets the video loopState, which should be an int cast from the enum VideoLoopType.LOOP\_NONE, VideoLoopType.LOOP\_NORMAL, or VideoLoopType.LOOP\_QUEUE. LOOP\_NONE means that the video will pause after it finishes playing once. LOOP\_NORMAL means the video will loop continuously. (No callbacks to videoEnd() are generated in LOOP\_NORMAL mode) LOOP\_QUEUE means the video will play through the end of the queue, and loop to the beginning.

**void setSpeed(IntPtr videoInstance, float speed);**

Sets the video rate of playback. You can set this negative, but please note that the queue list will not play backward in time.

**void setFrame(IntPtr videoInstance, int frame);**

Set the frame to a specific frame.

**void setPosition(IntPtr videoInstance, float videoPosition);**

Sets the position of the timeline to videoPosition, which should have a max value returned from getDuration()

### **VideoManagerSimpleGUI.cs**

Simple demo that lets you view GUI controls of each VideoInstance in the scene.

### ***VideoObject.cs***

This class interacts with instance-specific video methods.

**public VideoObject(string n);**

Constructor that takes in a filename. It stores the name, and creates a videoInstance.

**public VideoObject();**

Constructor that lets you use a VideoObject to simply store properties.

**public void initFrom(VideoObject copy);**

Essentially a deep copy method- lets you initialize a VideoObject from the properties of another.

**public void setVideoName(string myName);**

**public string getVideoName();**  
public interface for the videoName;

**public IntPtr getVideoInstance();**  
public interface for the videoInstance;

**public void destroyVideoObject();**  
public destructor for the VideoObject

**public void preloadVideo();**  
After the VideoObject has been created with the string constructor, you may call this to preload the video.

**public void setVideoLoaded(bool myLoaded);**  
**public bool getIsVideoLoaded();**  
public interface for determining if the video is loaded properly.

**public void setVideoLoopType(VideoObject.VideoLoopType loopType);**  
**public VideoLoopType getVideoLoopType();**  
public interface to the videoType

**public void setVideoErrorLoading(bool myError);**  
**public bool getErrorVideoLoading();**  
public interface to error checker

**public void setVideoSpeed(float myspeed);**  
**public float getVideoSpeed();**  
public interface for video speed control

**public void setVideoVolume(float myVolume);**  
**public float getVideoVolume();**  
public interface for video volume control

**public void setVideoWidthAndHeight();**  
only needs to be set once, after videoLoaded callback  
**public int getVideoWidth();**  
**public int getVideoHeight();**  
public interface for video width/height  
**public void setVideoTotalFrames();**  
only needs to be set once, after videoLoaded callback  
**public int getVideoTotalFrames();**  
public interface to get video's total frameCount

**public void setVideoDuration();**  
only needs to be set once, after videoLoaded callback  
**public int getVideoDuration();**  
public interface to get video's duration

**public void setVideoProperties();**

only needs to be set once, after videoLoaded callback. wrapper for  
setVideoWidthAndHeight(), setVideoTotalFrames(), and setVideoDuration()

**public void setVideoPaused(bool myIsPaused);**

**public bool getVideoPaused();**

public interface for setting video pause state

**public void setVideoPosition(float myPos);**

**public float getVideoPosition();**

public interface for video position

**public void setVideoFrame(int myPos);**

**public int getVideoFrame();**

public interface for setting video frame

**public void initializeProperties();**

**public void drawGUI();**

**public void drawLoopState();**

**public void drawTimeline(bool drawTime);**

**public void drawCurrentlyPlayingDetails();**

**public void drawIsPaused();**

**public void drawIsPausedEditor();**

**public void drawSpeed();**

**public void drawVolume();**

gui drawing methods

## **VideoObject.cs Native Plugin Methods**

**IntPtr createVideoInstance();**

Creates a video instance and returns the pointer- one created for each movie loaded.

**killVideoInstance(IntPtr videoInstance);**

kills the video instance (only use on quit or object-disable)

**void loadVideo(IntPtr videoInstance, string videoPath, bool autoPlay);**

Takes in video instance returned from createVideoInstance, the filename, and whether the video isPaused on load or not.

**float getSpeed(IntPtr videoInstance);**

Gets the current playback speed at which the video is set.

**float getPosition(IntPtr videoInstance);**

position of video. Position divided by duration goes from 0 to 1

**float getDuration(IntPtr videoInstance);**

duration of video.

**int getCurrentFrame(IntPtr videoInstance);**

Gets the current frame of the video.

**int getTotalNumFrames(IntPtr videoInstance);**

Gets the total number of frames in the video.

**int getLoopState(IntPtr videoInstance);**

Gets the loopState of the video.

**bool getIsMovieDone(IntPtr videoInstance);**

Gets whether the movie has ended or not. Not very useful anymore now that callbacks are integrated into the plugin.

**int getWidth(IntPtr videoInstance);**

Gets the video's internal width.

**int getHeight(IntPtr videoInstance);**

Gets the video's internal height.

**bool isLoaded(IntPtr videoInstance);**

Asks if the movie is loaded- not very useful now with callbacks.

**bool isLoading(IntPtr videoInstance);**

Asks if the movie is loading – not very useful now with callbacks.

**void setPaused(IntPtr videoInstance, bool isPaused);**

Sets whether the video should be playing or paused.

**void setVolume(IntPtr videoInstance, float myVol);**

Sets the volume from 0.0f to 1.0f.

**void setLoopState(IntPtr videoInstance, int loopState);**

Sets the video loopState, which should be an int cast from the enum VideoLoopType.LOOP\_NONE, VideoLoopType.LOOP\_NORMAL, or VideoLoopType.LOOP\_QUEUE. LOOP\_NONE means that the video will pause after it finishes playing once. LOOP\_NORMAL means the video will loop continuously. (No callbacks to videoEnd() are generated in LOOP\_NORMAL mode) LOOP\_QUEUE means the video will play through the end of the queue, and loop to the beginning.

**void setSpeed(IntPtr videoInstance, float speed);**

Sets the video rate of playback. You can set this negative, but please note that the queue list will not play backward in time.

**void setFrame(IntPtr videoInstance, int frame);**

Set the frame to a specific frame.

**void setPosition(IntPtr videoInstance, float videoPosition);**

Sets the position of the timeline to videoPosition, which should have a max value returned from getDuration()

## Comments

Please send questions/feature requests/bug reports to [brian@chasalow.com](mailto:brian@chasalow.com)

If sending a bug report, if it was a result of a crash, please include the .crash report located in /Library/Logs/DiagnosticReports/ , operating system version, and a brief description of what you may have been doing to cause the crash.

## Credits

Thanks to Anton Marini, Aras Pranckevičius and Rob Ramirez for helping to make this possible, and to James George for a portion of code that helped inspire this project. Many thanks to Deborah Johnson, my creative collaborator at CandyStations for logo design. ([www.candystations.com](http://www.candystations.com)) Landscape photographs displayed as promotional content are used, with permission, from Braden King's multimedia film/performance 'HERE [ The Story Sleeps ]' <http://herefilm.info/>

A portion of code was inspired by James George's fantastic ofxQTKitVideoPlayer addon for OpenFrameworks. The default Asset Store license applies to all files distributed with VideoTexture Pro.

The license for ofxQTKitVideoPlayer is below:

```
* Created by James George, http://www.jamesgeorge.org
* over a long period of time for a few different projects in
collaboration with
* FlightPhase http://www.flightphase.com
* and the rockwell group lab http://lab.rockwellgroup.com
*
*****
*
* Permission is hereby granted, free of charge, to any person
* obtaining a copy of this software and associated documentation
* files (the "Software"), to deal in the Software without
* restriction, including without limitation the rights to use,
* copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the
```

\* Software is furnished to do so, subject to the following  
\* conditions:  
\*  
\* The above copyright notice and this permission notice shall be  
\* included in all copies or substantial portions of the Software.  
\*  
\* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,  
\* EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES  
\* OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND  
\* NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT  
\* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,  
\* WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING  
\* FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR  
\* OTHER DEALINGS IN THE SOFTWARE.  
\*  
\* -----  
\*