IOS OpenGL ES 2 Movie Library.

The OpenGL Movie Library reads mov/mp4 files and updates a texture with frames of the movie synchronised to the movie sound track. Functions and the OpenGlMovieDelegate protocol are defined in OpenGLMovie.h, see comments within this file. Also see the movieES2 folder for an example implementation.

Basic Usage

Basic usage to play a movie once, is an initialisation and then a call in the render loop within an EAGLView. Apple seems to change the way they like to do init and rendering, so see the example implementation, which should help you restructure to the way you like to do it!

```
#import "OpenGLMovie.h"

-(void) setupMovie {

OpenGLMovieInit(@"trailer_480p.mov");

}

- (void)glkView:(GLKView *)view drawInRect:(CGRect)rect
{
    glClearColor(0.65f, 0.65f, 0.65f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glUseProgram(_program);

    // Movie stuff
    const GLfloat elementVerticies[] = {
        -1.0, 1.0, 0.0,     // Top left
        -1.0, -1.0, 0.0,    // Bottom left
        1.0, -1.0, 0.0,     // Bottom right
        1.0, 1.0, 0.0       // Top right
    };

    const GLushort standardIndices[] = {
        0,1,2,3
    };

    // set the position vertex attribute with our triangle's vertices
    glVertexAttribPointer(ATTRIB_POSITION, 3, GL_FLOAT, false, 0,
elementVerticies);
    glEnableVertexAttribArray(ATTRIB_POSITION);

    GLKMatrix4 baseModelViewMatrix = GLKMatrix4MakeTranslation(0.0f, 0.0f,
-6.0f);
    // set the texture uniform
    glActiveTexture(GL_TEXTURE0);// choose texture unit
    if(OpenGLMovieUpdateI()) {// binds texture -- RGBA
       float width=(float) OpenGLMovieWidthI();
       float height=(float) OpenGLMovieHeight();
       float texHeight=(float) OpenGLMovieTextureHeight(i);
       float texWidth=(float) OpenGLMovieTextureWidthIndex();

       const GLfloat movieTextureCoordinates[] = {
           0.0, 0.0,
           0.0, height/texHeight,
           width/texWidth, height/texHeight,
           width/texWidth, 0.0
       };
       float ratio=width/height;
        GLKMatrix4  modelViewMatrix =
GLKMatrix4MakeScale(modelViewMatrix,ratio,1.0f,1.0f);
        modelViewMatrix = GLKMatrix4Multiply(baseModelViewMatrix,
modelViewMatrix);
        _modelViewProjectionMatrix = GLKMatrix4Multiply(projectionMatrix,
modelViewMatrix);
        glUniformMatrix4fv(uniformMvp, 1, GL_FALSE, _modelViewProjectionMatrix.m
```

```
);
        glUniform1i(uniformTexture, index);

       // set the texture coords
       glVertexAttribPointer(ATTRIB_TEXCOORD, 2, GL_FLOAT, 0, 0,
movieTextureCoordinates);
       glEnableVertexAttribArray(ATTRIB_TEXCOORD);

       glDrawElements(GL_TRIANGLE_FAN, 4, GL_UNSIGNED_SHORT, standardIndices);
    }
}
```

When the movie has finished the movie texture will no longer be displayed. Note the initialisation is asynchronous so the width, and height of both, the movie itself and it's texture are not guaranteed to be immediately available. The OpenGLMovieUpdate will only return true when the movie has been properly initialise. However a more efficient approach it to set the texture coordinates only once by using the delegate's movieReady call to determine when the texture and movie sizes parameters are valid. See below for more details.

Looping or Playing a Sequence of Movies.

When the movie has finished playing it will attempt to send a movieFinished call to a delegate which conforms to the OpenGLMovieDelegate protocol defined in OpenGLMovie.h. To receive this call the user must set a delegate on initialisation.

```
  MyOpenGLMovieDelegate *finishedMovieDo=[[[MyOpenGLMovieDelegate alloc] init]
autorelease]; // Do this first to allow delegate to catch any errors
  OpenGLMovieSetDelegate(finishedMovieDo);
  OpenGLMovieInit(@"trailer_480p.mov");
  OpenGLMovieDisplayOnPause(true);//if movie paused still display texture
```

A simple delegate to loop the movie would be defined as ,

```
@interface MyOpenGLMovieDelegate : NSObject<OpenGLMovieDelegate> {}
@end

@implementation MyOpenGLMovieDelegate
  - (void) movieFinished {
      NSLog(@"finished");
       OpenGLMovieRewind();
  }
@end
```

To start a different movie it should be initialised as before, noting that a new delegate is not required as the existing delegate remains in place.

```
@implementation MyOpenGLMovieDelegate
  - (void) movieFinished {
      NSLog(@"finished");
      OpenGLMovieInit(@"movie_480p.mov");
  }
@end
```

Detecting When a Movie is Ready to Play.

It is possible as we have seen above to play a movie without directly determining when it is ready to play. The OpenGLMovieUpdate function returns false when the movie is not ready to play. OpenGLMovieUpdate will also return false when paused without the OpenGLMovieDisplayOnPause set true so that when finished a movie can vanish from view.

To avoid repeatedly recreating the movie texture coordinates the user can define an optional OpenGLMovieDelegate selector movieReady. This selector is triggered when the movie and texture parameters are valid.

```
@interface MyOpenGLMovieDelegate : NSObject<OpenGLMovieDelegate> {
    - (void) movieReady;
}
@end

@implementation MyOpenGLMovieDelegate
  - (void) movieReady {
      NSLog(@"Ready");
      DoInitOfTextureCoords();
  }
@end
```

Pause, Resume.

A playing movie can be paused using the OpenGLMoviePause function. This will pause the movie and stop the sound track. Note, in practice this increments a pauseLevel, so that for every pause a corresponding resume is required to decrement the pauseLevel. Only when pauseLevel is zero can the movie actually play/resume. The state of pause can be determined using the OpenGLMoviePauseLevel().

If OpenGLMovieDisplayOnPause was set true then the texture will be displayed with the frame that was current when pause was called, otherwise it will not be displayed. Internally this means where OpenGLMovieDisplayOnPause was set true a glBindTexture(GL_TEXTURE_2D, texture) call is make.

A movie is resumed using the OpenGLMovieResume function. This restarts the sound track and restarts the update of textures to match the sound track.

When there is a hardware pause (application goes into the background) the movie must be paused and on restarted must be resumed. For example in the AppDelegate the following could be used:

```
bool _hasResigned =false;

- (void)applicationWillResignActive:(UIApplication *)application {
  // depending on your system you might require [glView stopAnimation];
  [glView pauseMovies];
  _hasResigned=true;
}

- (void)applicationDidBecomeActive:(UIApplication *)application {
  if(_hasResigned) {
    // depending on your system you might require [glView startAnimation];
    [glView resumeMovies];
    _hasResigned=false;
  }
}
```

Seek.

To start movie at other than beginning the OpenGLMovieAtTime(@"move.mov",seekTime) is used, where seekTime is a float determining the number of seconds in to the movie that playback should start. If seekTime is longer than the duration of the movie it will not start. A movie that has been initialised with a seek time will restart at that seek time after a rewind operation.

Volume

The volume of a movie track can be adjusted using the OpenGLMovieVolume(volume) function, where volume is a float taking a value between 0.0f and 1.0f. However it is not recommended to use this function unless you have more than one movie playing or another sound playing and you want to adjust the relative volumes of the sound tracks.

Fast Textures

To enable the use of hardware texture copying the application must pass in the EAGLContext that is using the textures. A call of

```
OpenGLMovieFastTexture(_context); // use hardware texture copying
```

Will enable hardware copying. Without this call the library will use the CPU to copy textures. Hardware copying is recommended as it is much faster, especially for higher definition movies. With fast textures the ipad3 can happily play a 1080p (1920×1080) movie with plenty of cpu to spare.

Streaming Movies

CHECK APPLES LATEST POLICY ON STREAMING OVER 3GS CONNECTIONS IF YOU ARE SUBMITTING A STREAMING APP TO THE APPSTORE

This is a new feature and the interface is likely to expand as the feature evolves.

Note at present only one streaming movie is permitted and it will always be at index 0.

Basic usage is the same as for file based movies except that the command,

```
OpenGLMovieInitStream(@"http://mirrorblender.../big_buck_bunny_480p_h264.mov");
```

is used. With the http address of the movie. Note the movie must be a “faststart” movie. That is one designed to be streamed over the internet.

There are a number of optional delegate methods that can be used with streaming. This most important are,

```
// called when a stream is starting to play for first time
-(void) streamReady;

// called when a stream movie that is playing is paused/resumed because data is
// not available/become available again
// Only called if a frame was required and could not be delivered
// if movie is on pause already this is not called
-(void) streamPause:(BOOL) isPaused;
```

The movieReady delegate method is called but at this point the movie is not ready to play, but movie and texture size information is available. The movie can only be played after a few second worth of movie have been downloaded. At which point the delegate method streamReady will be called. If you have a good connection and fast download the next call that will received is the movieFinished call. But if the download rate cannot supply enough data to “feed” the movie the delegate method streamPause: isPaused is called, with isPause true. You may use this call to start a stall/waiting indicator. When the movie gets enough data to continue the delegate method streamPause: isPaused is called again. This time with isPause false. These methods can be called multiple times during playback.

Note it is not possible to seek to a position in a streaming movie. Once the movie has finished, it is possible to call the OpenGLMovieRewind function. The movie is now a file based movie in the app tmp area and by assessing the movie by it's file name it is possible to use it as a file based movie.

Note this tmp area can be delete by the system to free up space, so when restarting the application the movie file may have been deleted.

Playing Multiple Movies

Up to four movies can be played simultaneously. When using multiple movies the OpenGLMovieXXXIndex(index,....) functions are used. The OpenGLMovieXXX(...) functions use index 0, so that OpenGLMovieInitIndex(0,@”movie.mov”) is the equivalent to OpenGLMovieInit(@”movie.mov”).

The OpenGLMovieDelegate protocol has support for multiple movies with optional methods,

```
-(void) movieFinished: (int) index;
-(void) movieReady: (int) index;
```

This allows a single delegate such as the viewController to handle multiple movie finishedMovie calls. For example,

```
@interface MovieTestViewController:GLKViewController<OpenGLMovieDelegate> {
}

- (void) movieFinished: (int) index {
 static int frame[4]={0,0,0,0};
    NSLog(@"finished %d",index);
    if(index == 0) {
        frame[index]++;
        if(frame[index] %2 == 1) {
            if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad) {
                OpenGLMovieInitIndex(index,@"sintel_trailer-1080p.mp4");
            }
            else {
                OpenGLMovieInitIndex(index,@"sintel_trailer-720p.mp4");
            }
        } else {
            OpenGLMovieInitIndex(index,@"trailer_480p.mov");
        }
    }
    if(index == 3 || index == 1 || index == 2) {
        OpenGLMovieRewindIndex(index);
    }
}
```

Example Shader

The texture generated by the Movie Library is a standard RGBA texture. So in shader code it can be treated the same as any other texture.

```
// Standard Fragment Shader
varying mediump vec2 v_texcoord;
uniform sampler2D texture;

void main()
{
     gl_FragColor = texture2D(texture, v_texcoord);
}

// Standard Fragment Shader
attribute vec4 position;
attribute vec2 texcoord;
uniform mat4 mvp;

varying mediump vec2 v_texcoord;

void main()
{
    gl_Position = mvp * position;
     v_texcoord=texcoord;
}
```

An example shader is included in the moveES2 project.