MobiMesh Tutorial

Table of Contents

[Introduction 1](#_Toc297204406)

[MobiMesh xCode Project Organization 1](#_Toc297204407)

[TVPMRenderer Walkthrough 2](#_Toc297204408)

[Creating an OpenGL ES Application 2](#_Toc297204409)

[Setting up Depth Buffer 3](#_Toc297204410)

[Accessing Framebuffer Dimensions 4](#_Toc297204411)

[Truly View-dependent Progressive Mesh *(.tvpm)* Rendering 4](#_Toc297204412)

[Releasing Objects 7](#_Toc297204413)

[Rendering the Mesh 8](#_Toc297204414)

[Running the Application 8](#_Toc297204415)

[Extra Notes and Conclusion 8](#_Toc297204416)

# Introduction

This document provides a brief guide on how to use libMobiMesh to render the proposed *truly view-dependent meshes* *(.tvpm)* on the iPhone. The library supports both the standard *progressive mesh* *(.pm)* and the *view dependent progressive mesh (.vpm)* types as well. Rendering these other formats is similar to rendering the *.tvpm* format and will not be covered in this tutorial.

The main classes used when rendering *.tvpm* meshes are:

* GLMeshRenderer
  + GLMeshRendererES1 (when using OpenGL ES 1.1)
  + GLMeshRendererES2 (when using OpenGL ES 2.0)
* GLTvpmMeshRenderer
* TrulyViewDependentPM
  + TrulyViewDependentPM32 (for *.tvpm* 32-bit vertex ID meshes)
  + TrulyViewDependentPM64 (for *.tvpm64* 64-bit vertex ID meshes)

This tutorial covers only the basics of *.tvpm* rendering using libMobiMesh. It provides a sample application of the *thai.tvpm* mesh rendered and rotating constantly at 1 degree per frame about the y-axis.

# MobiMesh xCode Project Organization

The relevant files are stored in the MobiMesh xCode project. This project consists of four targets:

* MobiMesh: An iPhone application that uses the libMobiMesh library to render meshes. The relevant files can be found in the Classes group. Nib files for the interface are stored in the Resources group.
* libMobiMesh: The library itself. The relevant files are stored in the libMobiMesh group.
* MobiMeshIPad: Similar to MobiMesh but for the iPad. Both targets share the same source code and view controllers but have different nib files. Nib files for this target can be found in the MobiMeshIPad group.
* TVPMRenderer: Sample code that will be used for this tutorial. Contains the most basic logic required for rendering the *thai.tvpm* mesh on an iPhone using libMobiMesh.

Mesh files can also be found in the Resources group. The mesh extension indicates the mesh format used. We will leave it to the reader to setup the each target’s bundle resources, files and dependencies accordingly when following the walkthrough.

# TVPMRenderer Walkthrough

This section gives a step-by-step walkthrough on the TVPMRenderer target, from creating the target to compiling and running it on the simulator.

## Creating an OpenGL ES Application

The TVPMRenderer was created using xCode’s OpenGL ES application template, which provides an easy way to set up the necessary rendering parameters for an OpenGL ES application.

1. Select the project MobiMesh and add a target.
2. Under “iOS”, select “Application”, and then select the “OpenGL ES Application” template.

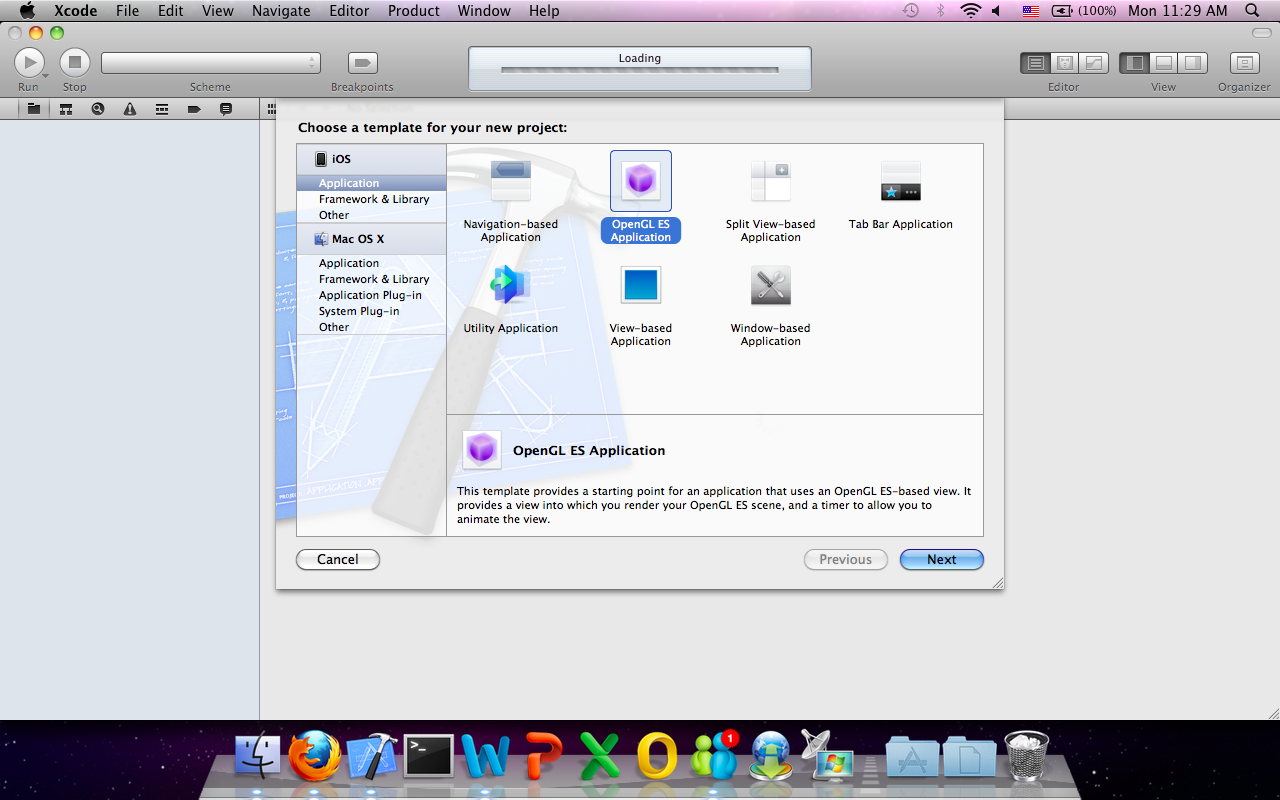


Figure Creating a new OpenGL ES Application

1. Click “Next”.
2. Enter the “Product Name” as “TVPMRenderer”.
3. Leave “Company Identifier” blank.
4. For “Device Family”, select “iPhone”.
5. Click “Create”.

The target will be created. Source files should also be found in the group TVPMRenderer. The following source files should be found in that group:

* TVPMRendererAppDelegate.h
* TVPMRendererDelegate.m
* EAGLView.h
* EAGLView.m
* TVPMRendererViewController.h
* TVPMRendererViewController.m

1. Rename the files to “TVPMRendererDelegate.mm” and “TVPMRendererViewController.mm” respectively to let the compiler know that these files will contain C++ code (from MobiMesh) as well.

## Setting up Depth Buffer

The template OpenGL ES application creates a frame and color render buffer but does not contain a depth buffer. However, having a depth buffer helps in resolving distances between two nearby objects so that they do not randomly display through each other. This greatly improves the perceived quality of the mesh. The following steps describe how to set up the depth buffer. The creation of the depth buffer is done in the EAGLView.h file.

1. In EAGLView.h declare the depth render buffer, depthRenderbuffer.

|  |
| --- |
| @interface EAGLView : UIView {  @private  // The pixel dimensions of the CAEAGLLayer.  GLint framebufferWidth;  GLint framebufferHeight;    // The OpenGL ES names for the framebuffer and renderbuffer used to render  // to this view.  GLuint defaultFramebuffer, colorRenderbuffer;  // \*\*\* MobiMesh Tutorial: For storing depth renderbuffer  GLuint depthRenderbuffer;  } |

Figure EAGLView.h Declaring a depth buffer.

1. Initialize the depth render buffer in EAGLView.m createFramebuffer method.

|  |
| --- |
| - (void)createFramebuffer  {  if (context && !defaultFramebuffer)  {  // Existing auto generated code ...  glFramebufferRenderbuffer(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0,  GL\_RENDERBUFFER, colorRenderbuffer);  // \*\*\* MobiMesh Tutorial: Create depth renderbuffer  glGenRenderbuffers(1, &depthRenderbuffer);  glBindRenderbuffer(GL\_RENDERBUFFER, depthRenderbuffer);  glRenderbufferStorage(GL\_RENDERBUFFER\_OES, GL\_DEPTH\_COMPONENT16,  framebufferWidth, framebufferHeight);  glFramebufferRenderbuffer(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT,  GL\_RENDERBUFFER, depthRenderbuffer);    // Continue existing auto generated code …  if (glCheckFramebufferStatus(GL\_FRAMEBUFFER) != GL\_FRAMEBUFFER\_COMPLETE) {  NSLog(@"Failed to make complete framebuffer object %x",  glCheckFramebufferStatus(GL\_FRAMEBUFFER));  }  }  } |

Figure EAGLView.m createFramebuffer Initializing the depth buffer.

1. Free the depth render buffer in EAGLView.m deleteFramebuffer method.

|  |
| --- |
| - (void)deleteFramebuffer  {  if (context) {  // Existing auto generated code ...  if (colorRenderbuffer) {  glDeleteRenderbuffers(1, &colorRenderbuffer);  colorRenderbuffer = 0;  }  // \*\*\* MobiMesh Tutorial: free depth buffer  if (depthRenderbuffer) {  glDeleteRenderbuffers(1, &depthRenderbuffer);  depthRenderbuffer = 0;  }  }  } |

Figure EAGLView.m deleteFramebuffer Freeing the depth buffer.

## Accessing Framebuffer Dimensions

The renderer classes provided by libMobiMesh require the viewport dimensions which are stored in the EAGLView class, we need to expose these parameters so that the view controllers, which will be the one creating the renderers, can access them. XCode’s template already provides and populates these variables when the color renderbuffer is created, so we only need to expose them as properties as follows.

1. In EAGLView.h, declare the framebufferWidth and framebufferHeight variables as properties of the EAGLView interface.

|  |
| --- |
| // \*\*\* MobiMesh Tutorial: Framebuffer size for view controller to access  @property (nonatomic, readonly) GLint framebufferWidth;  @property (nonatomic, readonly) GLint framebufferHeight; |

Figure EAGLView.h Exposing framebuffer dimensions as interface properties.

1. In EAGLView.m, synthesize these properties to auto-generate the getters and setters.

|  |
| --- |
| @synthesize context;  // \*\*\* MobiMesh Tutorial: Framebuffer dimensions for view controller to access  @synthesize framebufferWidth, framebufferHeight; |

Figure EAGLView.h Synthesize framebuffer dimensions.

## Truly View-dependent Progressive Mesh *(.tvpm)* Rendering

The loading of the mesh and creation of renderers are done in TVPMRendererViewController.h and TVPMViewController.mm. We begin by declaring the necessary variables in TVPMRendererViewController.h, then initializing and using them in TVPMRendererViewController.mm.

1. In TVPMRendererViewController.h, import the necessary headers.

|  |
| --- |
| #import “GLMeshRenderer.h”  #import “GLTvpmMeshRenderer.h”  #import “TrulyViewDependentPM.h” |

Figure TVPMRendererViewController.h Importing required header files.

1. In TVPMRendererViewController.h, in TVPMRendererViewController’s interface, declare the necessary data members for rendering TVPM mesh.

|  |
| --- |
| // \*\*\* MobiMesh Tutorial: Declare necessary renderer objects  MobiMesh::GLMeshRenderer\* renderer;  MobiMesh::GLTvpmMeshRenderer\* tvpmRenderer;  MobiMesh::TrulyViewDependentPM\* mesh; |

Figure TVPMRendererViewController.h Declaring renderer and mesh data members.

1. In TVPMRendererViewController.mm, import the necessary libMobiMesh headers again. These headers are different from TVPMRendererViewController.h’s because they contain classes that are concrete subclasses of those imported in TVPMRendererViewController.h

|  |
| --- |
| #import “GLMeshRendererES1.h”  #import “GLMeshRendererES2.h”  #import “TrulyViewDependentPM32.h”  #import “FileReadErrorException.h”  #import “ShaderCompileException.h”  #import “ProgramLinkingException.h”  #import “InvalidGLProgramException.h” |

Figure TVPMRendererViewController.mm Including necessary header files of relevant concrete classes.

1. In TVPMRendererViewController.mm, the awakeFromNib method contains code to create the renderers and mesh objects declared in TVPMRendererViewController.h. Note that the version of OpenGL ES to use is determined by the context’s initialization which has already been done in the sample code. In this awakeFromNib method, we first create the mesh and for this tutorial, we use *thai.tvpm*. Error handling is also shown here. In the case of this tutorial, we simply log the exception’s error message and exit with an error code of 1.

|  |
| --- |
| // \*\*\* MobiMesh Tutorial: Create the mesh  NSString \*meshPath = [[NSBundle mainBundle] pathForResource:@"thai" ofType:@"tvpm"];  try  {  mesh = new MobiMesh::TrulyViewDependentPM32(  [meshPath cStringUsingEncoding:NSASCIIStringEncoding]);  }  catch (MobiMesh::FileReadErrorException& ex)  {  NSLog(@"%s", ex.what());  exit(1);  }  catch (MobiMesh::InvalidFileFormatException& ex)  {  NSLog(@"%s", ex.what());  exit(1);  } |

Figure TVPMRendererViewController.mm awakeFromNib Loading thai.tvpm.

1. Depending on the version of OpenGL ES the context is initialized with, we create the specific renderer (GLMeshRendererES1 or GLMeshRendererES2) instance accordingly. Suppose the context is using OpenGL ES 2.0, then we have to supply the vertex and fragment shaders in the renderer creation as follows. Shader compilation takes place when the renderer object is constructed but can result in errors. We should remember to catch the appropriate exceptions. In this tutorial, we simply log the exception’s error message and exit with code 1 when an error occurs.

|  |
| --- |
| if ([context API] == kEAGLRenderingAPIOpenGLES2)  {  // \*\*\* MobiMesh Tutorial: Load shaders for renderer if using ES2  NSString \*vertexShaderFile = [[NSBundle mainBundle] pathForResource:@"Shader" ofType: @"vsh"];  NSString \*fragmentShaderFile = [[NSBundle mainBundle] pathForResource: @"Shader" ofType: @"fsh"];  NSString \* vertexShaderSource =  [NSString stringWithContentsOfFile:vertexShaderFile encoding:NSUTF8StringEncoding error:nil];  NSString \* fragmentShaderSource =  [NSString stringWithContentsOfFile:fragmentShaderFile encoding:NSUTF8StringEncoding error:nil];  if (!fragmentShaderSource || !vertexShaderSource)  {  NSLog(@"Failed to load shaders");  exit(1);  }  // \*\*\* MobiMesh Tutorial: Create the renderer and exit on failure.  try  {  renderer = new MobiMesh::GLMeshRendererES2(  [vertexShaderSource cStringUsingEncoding:NSUTF8StringEncoding],  [fragmentShaderSource cStringUsingEncoding:NSUTF8StringEncoding],  [(EAGLView \*)self.view framebufferWidth],  [(EAGLView \*)self.view framebufferHeight]);  }  catch (MobiMesh::ShaderCompileException& ex)  {  NSLog(@"%s", ex.what());  exit(1);  }  catch (MobiMesh::ProgramLinkingException& ex)  {  NSLog(@"%s", ex.what());  exit(1);  }  catch (MobiMesh::InvalidGLProgramException& ex)  {  NSLog(@"%s", ex.what());  exit(1);  }  } |

Figure TVPMRendererViewController.mm awakeFromNib Creating GLMeshRendererES2 object.

Notice that EAGLView’s framebuffer dimension properties that we exposed in the previous section are being supplied as the renderer’s viewport dimensions. Also, since shader compilation is done when the renderer is created, xCode’s template shader compilation and validation code in TVPMRendererViewController.mm can be removed.

1. If we are using OpenGL ES 1.1, then the shaders need not be supplied. The relevant renderer object to be created is GLMeshRendererES1. The lighting and material for OpenGL ES 1.1 must also be set. (For OpenGL ES 2.0, this is done in the shader.)

|  |
| --- |
| else  {  // \*\*\* MobiMesh Tutorial: Create ES1 renderer if using ES1  renderer = new MobiMesh::GLMeshRendererES1(  [(EAGLView \*)self.view framebufferWidth],  [(EAGLView \*)self.view framebufferHeight]);  // \*\*\* MobiMesh Tutorial: Setup lighting and material  GLfloat mat[4];  mat[0]=0.2125; mat[1]=0.1275; mat[2]=0.054; mat[3]=1.0;  glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, mat);  mat[0]=0.714; mat[1]=0.4284; mat[2]=0.18144;  glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, mat);  mat[0]=0.393548; mat[1]=0.271906; mat[2]=0.166721;  GLfloat shine = 0.2;  glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, mat);  glMaterialf (GL\_FRONT\_AND\_BACK, GL\_SHININESS, shine \* 128.0);  GLfloat light\_position [] = { 0.0, 0.0, 1.0, 0.0};  glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);  glEnable(GL\_LIGHTING);  glEnable(GL\_LIGHT0);  glEnable(GL\_NORMALIZE);  } |

Figure TVPMRendererViewController.mm awakeFromNib Creating GLMeshRendererES1 object and setting up material and lighting.

1. With the GLMeshRenderer object created, we can now supply it to the GLTvpmRenderer object in its constructor.

|  |
| --- |
| // \*\*\* MobiMesh Tutorial: Create the TVPM renderer  tvpmRenderer = new MobiMesh::GLTvpmMeshRenderer(\*renderer, mesh); |

Figure TVPMRendererViewController.mm awakeFromNib Creating the GLTvpmMeshRenderer object by supplying the renderer and mesh objects created from before.

1. All the objects have been initialized. Before we start rendering, we should enable depth testing and back face culling for better results. Depth testing will make use of the depth buffer that was created in EAGLView.

|  |
| --- |
| // \*\*\* MobiMesh Tutorial: Enable depth testing for more accurate rendering  glEnable(GL\_DEPTH\_TEST);  // \*\*\* MobiMesh Tutorial: Enable back face culling  glEnable(GL\_CULL\_FACE); |

Figure TVPMRendererViewController.mm awakeFromNib Enabling depth testing and back face culling.

## Releasing Objects

The creation of objects involved dynamic memory allocation with the new operator. This means that the objects must also be freed when the view controller is freed up. This can be done in TVPMRendererViewController.mm in the dealloc method as follows.

|  |
| --- |
| // \*\*\* MobiMesh Tutorial: Release renderers and mesh  if (tvpmRenderer)  delete tvpmRenderer;  if (renderer)  delete renderer;  if (mesh)  delete mesh; |

Figure TVPMRendererViewController.mm dealloc Freeing renderer and mesh objects.

Note that because the *GLTvmMeshRenderer tvmRenderer* object uses the *GLMeshRenderer renderer* and *TrulyViewDependentPM32 mesh* objects for rendering, we avoid releasing *renderer* and *mesh* before it. Likewise, the *GLMeshRenderer render* object might use *mesh* for rendering, so we do not release *mesh* until we have released *renderer*. Although changing the order in this particular function does not have any effect since these objects will not be accessed subsequently, it is good practice to conform to the object lifetime constraints imposed by the library. Besides, this order of releasing might become more significant in a multi-threaded application.

## Rendering the Mesh

We have covered the setup and tear down of the mesh and renderers. Rendering the mesh is all that remains. This is done in TVPMRendererViewController.mm in the drawFrame method which is called in every frame. For purposes of this tutorial, we will rotate the mesh by 1 degree about the y-axis in every frame, and render the mesh. The code for doing so is simply:

|  |
| --- |
| - (void)drawFrame  {  [(EAGLView \*)self.view setFramebuffer];  // \*\*\* MobiMesh Tutorial: Rotate by 1 degree about y-axis in each frame and render the mesh.  tvpmRenderer->rotate(1.0f, 0.0f, 1.0f, 0.0f);  renderer\_->render();    [(EAGLView \*)self.view presentFramebuffer];  } |

Figure TVPMRendererViewController.mm drawFrame Rotating the mesh in each frame and rendering it.

## Running the Application

The code is now complete. Select the TVPMRenderer scheme and click “Run”. You should see the Thai mesh rendered and rotating about its y-axis. Adaptation of the mesh to the current view also takes place in each frame. This adaptation is actually performed in the render() call, so the user of the library need not refine the mesh in code.



Figure TVPMRenderer application rendering *thai.tvpm* on the simulator.

# Extra Notes and Conclusion

This tutorial provided shows only the bare minimum of what libMobiMesh can do. It has not described how the equivalent PM and VPM meshes can be rendered. However, rendering these meshes is similar to rendering TVPM meshes because all renderers implement the IMesh interface. The only difference is in the relevant renderer and mesh classes used. For more sample code, one can also refer to our MobiMesh application (Project target: MobiMesh). For a description of other methods in the IMesh interface, refer to the full technical report or the Doxygen documentation.