Checklist for Building OpenGL ES Apps for iOS

The OpenGL ES specification defines a platform-neutral API for using GPU hardware to render graphics. Platforms implementing OpenGL ES provide a rendering context for executing OpenGL ES commands, framebuffers to hold rendering results, and one or more rendering destinations that present the contents of a framebuffer for display. In iOS, the `EAGLContext` class implements a rendering context. iOS provides only one type of framebuffer, the OpenGL ES framebuffer object, and the `GLKView` and `CAEAGLLayer` classes implement rendering destinations.

Building an OpenGL ES app in iOS requires several considerations, some of which are generic to OpenGL ES programming and some of which are specific to iOS. Follow this checklist and the detailed sections below to get started:

1. Determine which version(s) of OpenGL ES have the right feature set for your app, and create an OpenGL ES context.
2. Verify at runtime that the device supports the OpenGL ES capabilities you want to use.
3. Choose where to render your OpenGL ES content.
4. Make sure your app runs correctly in iOS.
5. Implement your rendering engine.
6. Use Xcode and Instruments to debug your OpenGL ES app and tune it for optimal performance .

Choosing Which OpenGL ES Versions to Support

Decide whether your app should support OpenGL ES 3.0, OpenGL ES 2.0, OpenGL ES 1.1, or multiple versions.

- OpenGL ES 3.0 is new in iOS 7. It adds a number of new features that enable higher performance, general-purpose GPU computing techniques, and more complex visual effects previously only possible on desktop-class hardware and game consoles.
- OpenGL ES 2.0 is the baseline profile for iOS devices, featuring a configurable graphics pipeline based on programmable shaders.
- OpenGL ES 1.1 provides only a basic fixed-function graphics pipeline and is available in iOS primarily for backward compatibility.

You should target the version or versions of OpenGL ES that support the features and devices most relevant to your app. To learn more about the OpenGL ES capabilities of iOS devices, read *iOS Device Compatibility Reference*.

To create contexts for the versions of OpenGL ES you plan to support, read Configuring OpenGL ES Contexts. To learn how your choice of OpenGL ES version relates to the rendering algorithms you might use in your app, read OpenGL ES Versions and Renderer Architecture.

Verifying OpenGL ES Capabilities

The *iOS Device Compatibility Reference* summarizes the capabilities and extensions available on shipping iOS devices. However, to allow your app to run on as many devices and iOS versions as possible, your app should always query the OpenGL ES implementation for its capabilities at

runtime.

To determine implementation specific limits such as the maximum texture size or maximum number of vertex attributes, look up the value for the corresponding token (such as `MAX_TEXTURE_SIZE` or `MAX_VERTEX_ATTRIBS`, as found in the `gl.h` header) using the appropriate `glGet` function for its data type.

To check for OpenGL ES 3.0 extensions, use the `glGetIntegerv` and `glGetStringi` functions as in the following code example:

```
BOOL CheckForExtension(NSString *searchName)

{

    // Create a set containing all extension names.

    // (For better performance, create the set only once and cache it for future
use.)

    int max = 0;

    glGetIntegerv(GL_NUM_EXTENSIONS, &max);

    NSMutableSet *extensions = [NSMutableSet set];

    for (int i = 0; i < max; i++) {

        [extensions addObject: @( (char *)glGetStringi(GL_EXTENSIONS, i) )];

    }

    return [extensions containsObject: searchName];

}
```

To check for OpenGL ES 1.1 and 2.0 extensions, call `glGetString(GL_EXTENSIONS)` to get a space-delimited list of all extension names.

Choosing a Rendering Destination

In iOS, a framebuffer object stores the results of drawing commands. (iOS does not implement window-system-provided framebuffers.) You can use the contents of a framebuffer object in multiple ways:

- The GLKit framework provides a view that draws OpenGL ES content and manages its own framebuffer object, and a view controller that supports animating OpenGL ES content. Use these classes to create full screen views or to fit your OpenGL ES content into a UIKit view hierarchy. To learn about these classes, read Drawing with OpenGL ES and GLKit.
- The `CAEAGLLayer` class provides a way to draw OpenGL ES content as part of a Core Animation layer composition. You must create your own framebuffer object when using this class.
- As with any OpenGL ES implementation, you can also use framebuffers for offscreen graphics processing or rendering to a texture for use elsewhere in the graphics pipeline. With OpenGL ES 3.0, offscreen buffers can be used in rendering algorithms that utilize multiple render targets.

To learn about rendering to an offscreen buffer, a texture, or a Core Animation layer, read Drawing to Other Rendering Destinations.

Integrating with iOS

iOS apps support multitasking by default, but handling this feature correctly in an OpenGL ES app requires additional consideration. Improper use of OpenGL ES can result in your app being killed by the system when in the background.

Many iOS devices include high-resolution displays, so your app should support multiple display sizes and resolutions.

To learn about supporting these and other iOS features, read Multitasking, High Resolution, and Other iOS Features.

Implementing a Rendering Engine

There are many possible strategies for designing your OpenGL ES drawing code, the full details of which are beyond the scope of this document. Many aspects of rendering engine design are generic to all implementations of OpenGL and OpenGL ES.

To learn about design considerations important for iOS devices, read OpenGL ES Design Guidelines and Concurrency and OpenGL ES.

Debugging and Profiling

Xcode and Instruments provide a number of tools for tracking down rendering problems and analyzing OpenGL ES performance in your app.

To learn more about solving problems and improving performance in your OpenGL ES app, read Tuning Your OpenGL ES App.