iOS Device Compatibility Reference

Developer

Contents

Tables

Introduction

iOS devices support a variety of features, including sensors, graphics processors and networking options. When designing your app, you need to decide what capabilities your app needs and which devices to support, because the capabilities of each kind of iOS device are different.

At a Glance

This document describes the details for each device in order to help you develop your app and choose devices to test on. The information contained here is current as of iOS 8.0, but it is subject to change in future hardware or software releases.

Device Compatibility Strings

Sometimes, your app is dependent on a specific hardware feature existing on a device. On iOS, you can declare these dependencies when you build your app. When the app is built, this compatibility information is used to prevent the app from being installed on a device it can't run on—and it can also be used by the App Store to prevent a customer from purchasing an app that doesn't work on their device.

Chapter: Device Compatibility (page 6)

Metal and OpenGL ES

When working with Metal and OpenGL ES, you often need to know the exact capabilities of the underlying hardware and the software that talks to it. Metal and OpenGL ES provide many built-in mechanisms for determining this information; this document summarizes the most important information and provides other information useful when creating Metal and OpenGL ES apps that run well on iOS devices.

Chapter: Hardware GPU Information (page 18)

How to Use This Document

Although this document provides important Metal and OpenGL ES hardware information, it is not definitive. If you are unfamiliar with OpenGL ES programming, consult the *OpenGL ES Programming Guide for iOS* to learn how to develop OpenGL ES apps on iOS. If you are unfamiliar with Metal programming, consult the *Metal Programming Guide* to learn how to develop Metal apps on iOS. To ensure compatibility with future devices and iOS versions, your app must always test the capabilities of the underlying Metal and OpenGL ES implementation at runtime, disabling any features that do not have the required support from iOS.

Device Compatibility

The information property list (`Info.plist`) file contains critical information about your app's configuration and must be included in your app bundle. Every new project you create in Xcode has a default `Info.plist` file configured with some basic information about your project. You can modify this file to specify additional configuration details for your app.

The `UIRequiredDeviceCapabilities` key lets you declare the hardware or specific capabilities that your app needs in order to run. All apps are required to have this key in their `Info.plist` file. The App Store uses the contents of this key to prevent users from downloading your app onto a device that cannot possibly run it. The tables in this chapter show all iOS devices and their capabilities.

Important: All device requirement changes must be made when you submit an update to your binary. You are permitted only to expand your device requirements. Submitting an update to your binary to restrict your device requirements is not permitted. You are unable to restrict device requirements because this action will keep customers who have previously downloaded your app from running new updates.

Important: If you require a capability listed in bold, you must build your app as a fat binary (armv6 and armv7) or require a minimum iOS version of 4.3 or later. See the individual device tables for a specific key requirement.

Declaring the Required Device Capabilities

The value of the `UIRequiredDeviceCapabilities` key is either an array or a dictionary that contains additional keys identifying features your app requires (or specifically prohibits). If you specify the value of the key using an array, the presence of a key indicates that the feature is required; the absence of a key indicates that the feature is not required and that the app can run without it. If you specify a dictionary instead, each key in the dictionary must have a Boolean value that indicates whether the feature is required or prohibited. A value of `true` indicates the feature is required and a value of `false` indicates that the feature must *not* be present on the device. If a given capability is optional for your app, do not include the corresponding key in the dictionary.

Table 1-1 lists the keys that you can include in the array or dictionary for the `UIRequiredDeviceCapabilities` key. You should include keys only for the features that your app absolutely requires. If your app can run without a specific feature, do not include the corresponding key.

Table 1-1 Dictionary keys for the `UIRequiredDeviceCapabilities` key

Key	**Description**	**Minimum iOS Required**
`accelerometer`	Include this key if your app requires (or specifically prohibits) the presence of accelerometers on the device. Apps use the Core Motion framework to receive accelerometer events. You do not need to include this key if your app detects only device orientation changes.	iOS 3.0
`armv6`	Include this key if your app is compiled only for the armv6 instruction set.	iOS 2.0
`armv7`	Include this key if your app is compiled only for the armv7 instruction set.	iOS 3.1
`auto-focus-camera`	Include this key if your app requires (or specifically prohibits) autofocus capabilities in the device's still camera. Although most developers should not need to include this key, you might include it if your app supports macro photography or requires sharper images in order to perform some sort of image processing.	iOS 3.0
`bluetooth-le`	Include this key if your app requires (or specifically prohibits) the presence of Bluetooth low-energy hardware on the device.	iOS 5.0
`camera-flash`	Include this key if your app requires (or specifically prohibits) the presence of a camera flash for taking pictures or shooting video. Apps use the `UIImagePicker-Controller` interface to control the enabling of this feature.	iOS 3.0
`front-facing-camera`	Include this key if your app requires (or specifically prohibits) the presence of a forward-facing camera. Apps use the `UIImagePickerController` interface to capture video from the device's camera.	iOS 3.0
`gamekit`	Include this key if your app requires (or specifically prohibits) Game Center.	iOS 4.1

Key	Description	Minimum iOS Required
`gps`	Include this key if your app requires (or specifically prohibits) the presence of GPS (or AGPS) hardware when tracking locations. (You should include this key only if you need the higher accuracy offered by GPS hardware.) If you include this key, you should also include the `location-services` key. You should require GPS only if your app needs location data more accurate than the cellular or Wi-fi radios might otherwise provide.	iOS 3.0
`gyroscope`	Include this key if your app requires (or specifically prohibits) the presence of a gyroscope on the device. Apps use the Core Motion framework to retrieve information from gyroscope hardware.	iOS 3.0
`healthkit`	Include this key if your app requires (or specifically prohibits) HealthKit.	iOS 8.0
`location-services`	Include this key if your app requires (or specifically prohibits) the ability to retrieve the device's current location using the Core Location framework. (This key refers to the general location services feature. If you specifically need GPS-level accuracy, you should also include the `gps` key.)	iOS 3.0
`magnetometer`	Include this key if your app requires (or specifically prohibits) the presence of magnetometer hardware. Apps use this hardware to receive heading-related events through the Core Location framework.	iOS 3.0
`metal`	Include this key if your app requires (or specifically prohibits) Metal.	iOS 8.0
`microphone`	Include this key if your app uses the built-in microphone or supports accessories that provide a microphone.	iOS 3.0
`opengles-1`	Include this key if your app requires (or specifically prohibits) the presence of the OpenGL ES 1.1 interfaces.	iOS 3.0
`opengles-2`	Include this key if your app requires (or specifically prohibits) the presence of the OpenGL ES 2.0 interfaces.	iOS 3.0

Key	**Description**	**Minimum iOS Required**
`opengles-3`	Include this key if your app requires (or specifically prohibits) the presence of the OpenGL ES 3.0 interfaces.	iOS 7.0
`peer-peer`	Include this key if your app requires (or specifically prohibits) peer-to-peer connectivity over a Bluetooth network.	iOS 3.1
`sms`	Include this key if your app requires (or specifically prohibits) the presence of the Messages app. You might require this feature if your app opens URLs with the `sms` scheme.	iOS 3.0
`still-camera`	Include this key if your app requires (or specifically prohibits) the presence of a camera on the device. Apps use the `UIImagePickerController` interface to capture images from the device's still camera.	iOS 3.0
`telephony`	Include this key if your app requires (or specifically prohibits) the presence of the Phone app. You might require this feature if your app opens URLs with the `tel` scheme.	iOS 3.0
`video-camera`	Include this key if your app requires (or specifically prohibits) the presence of a camera with video capabilities on the device. Apps use the `UIImagePickerController` interface to capture video from the device's camera.	iOS 3.0
`wifi`	Include this key if your app requires (or specifically prohibits) access to the networking features of the device.	iOS 3.0

For detailed information on how to create and edit property lists, see *Information Property List Key Reference* .

iPhone Devices

Table 1-2 and Table 1-3 (page 11) list the capabilities for iPhone devices.

Table 1-2 iPhone 4, iPhone 5, and iPhone 6 device compatibility

Compatibility	**iPhone 4**	**iPhone 4s**	**iPhone 5**	**iPhone 5c**	**iPhone 5s**	**iPhone 6**	**iPhone 6 Plus**
accelerometer	X	X	X	X	X	X	X
armv6	X	X	X	X	X	X	X
armv7	X	X	X	X	X	X	X
auto-focus-camera	X	X	X	X	X	X	X
bluetooth-le		X	X	X	X	X	X
camera-flash	X	X	X	X	X	X	X
front-facing-camera	X	X	X	X	X	X	X
gamekit	X	X	X	X	X	X	X
gps	X	X	X	X	X	X	X
gyroscope	X	X	X	X	X	X	X
healthkit		X	X	X	X	X	X
location-services	X	X	X	X	X	X	X
magnetometer	X	X	X	X	X	X	X
metal					X	X	X
microphone	X	X	X	X	X	X	X
opengles-1	X	X	X	X	X	X	X
opengles-2	X	X	X	X	X	X	X
opengles-3					X	X	X
peer-peer	X	X	X	X	X	X	X
sms	X	X	X	X	X	X	X
still-camera	X	X	X	X	X	X	X
telephony	X	X	X	X	X	X	X
video-camera	X	X	X	X	X	X	X

Compatibility	**iPhone 4**	**iPhone 4s**	**iPhone 5**	**iPhone 5c**	**iPhone 5s**	**iPhone 6**	**iPhone 6 Plus**
wifi	X	X	X	X	X	X	X

Table 1-3 iPhone and iPhone 3G device compatibility

Compatibility	**iPhone**	**iPhone 3G**	**iPhone 3GS**	**iPhone 3GS (China)**
accelerometer	X	X	X	X
armv6	X	X	X	X
armv7			X	X
auto-focus-camera			X	X
bluetooth-le				
camera-flash				
front-facing-camera				
gamekit			X	X
gps		X	X	X
gyroscope				
healthkit				
location-services	X	X	X	X
magnetometer			X	X
metal				
microphone	X	X	X	X
opengles-1	X	X	X	X
opengles-2			X	X
opengles-3				
peer-peer		X	X	X
sms	X	X	X	X

Compatibility	iPhone	iPhone 3G	iPhone 3GS	iPhone 3GS (China)
still-camera	X	X	X	X
telephony	X	X	X	X
video-camera			X	X
wifi	X	X	X	

iPad Devices

Table 1-4 and Table 1-5 (page 13), and Table 1-6 (page 15) list the capabilities for iPad devices.

Table 1-4 iPad (4th generation), and iPad Air device compatibility

Compatibility	iPad Wi-Fi (4th gen)	iPad Wi-Fi + Cellular (4th gen)	iPad Air Wi-Fi	iPad Air Wi-Fi + Cellular	iPad Air 2 Wi-Fi	iPad Air 2 Wi-Fi + Cellular
accelerometer	X	X	X	X	X	X
armv6	X	X	X	X	X	X
armv7	X	X	X	X	X	X
auto-focus-camera	X	X	X	X	X	X
bluetooth-le	X	X	X	X	X	X
camera-flash						
front-facing-camera	X	X	X	X	X	X
gamekit	X	X	X	X	X	X
gps		X		X		X
gyroscope	X	X	X	X	X	X
healthkit						
location-services	X	X	X	X	X	X

Compatibility	iPad Wi-Fi (4th gen)	iPad Wi-Fi + Cellular (4th gen)	iPad Air Wi-Fi	iPad Air Wi-Fi + Cellular	iPad Air 2 Wi-Fi	iPad Air 2 Wi-Fi + Cellular
magnetometer	X	X	X	X	X	X
metal			X	X	X	X
microphone	X	X	X	X	X	X
opengles-1	X	X	X	X	X	X
opengles-2	X	X	X	X	X	X
opengles-3			X	X	X	X
peer-peer	X	X	X	X	X	X
sms						
still-camera	X	X	X	X	X	X
telephony						
video-camera	X	X	X	X	X	X
wifi	X	X	X	X	X	X

Table 1-5 iPad mini device compatibility

Compatibility	iPad mini Wi-Fi	iPad mini Wi-Fi + Cellular	iPad mini with Retina display Wi-Fi	iPad mini with Retina display Wi-Fi + Cellular	iPad mini 3 Wi-Fi	iPad mini 3 Wi-Fi + Cellular
accelerometer	X	X	X	X	X	X
armv6	X	X	X	X	X	X
armv7	X	X	X	X	X	X
auto-focus-camera	X	X	X	X	X	X
bluetooth-le	X	X	X	X	X	X

Compatibility	iPad mini Wi-Fi	iPad mini Wi-Fi + Cellular	iPad mini with Retina display Wi-Fi	iPad mini with Retina display Wi-Fi + Cellular	iPad mini 3 Wi-Fi	iPad mini 3 Wi-Fi + Cellular
camera-flash						
front-facing-camera	X	X	X	X	X	X
gamekit	X	X	X	X	X	X
gps		X		X		X
gyroscope	X	X	X	X	X	X
healthkit						
location-services	X	X	X	X	X	X
magnetometer	X	X	X	X	X	X
metal			X	X	X	X
microphone	X	X	X	X	X	X
opengles-1	X	X	X	X	X	X
opengles-2	X	X	X	X	X	X
opengles-3			X	X	X	X
peer-peer	X	X	X	X	X	X
sms						
still-camera	X	X	X	X	X	X
telephony						
video-camera	X	X	X	X	X	X
wifi	X	X	X	X	X	X

Table 1-6 iPad 1, iPad 2, and iPad (3rd generation) device compatibility

Compatibility	**iPad Wi-Fi**	**iPad Wi-Fi + 3G**	**iPad 2 Wi-Fi**	**iPad 2 Wi-Fi + 3G**	**iPad Wi-Fi (3rd gen)**	**iPad Wi-Fi + Cellular (3rd gen)**
accelerometer	X	X	X	X	X	X
armv6	X	X	X	X	X	X
armv7	X	X	X	X	X	X
auto-focus-camera					X	X
bluetooth-le					X	X
camera-flash						
front-facing-camera			X	X	X	X
gamekit	X	X	X	X	X	X
gps		X		X		X
gyroscope			X	X	X	X
healthkit						
location-services	X	X	X	X	X	X
magnetometer	X	X	X	X	X	X
metal						
microphone	X	X	X	X	X	X
opengles-1	X	X	X	X	X	X
opengles-2	X	X	X	X	X	X
opengles-3						
peer-peer	X	X	X	X	X	X
sms						
still-camera			X	X	X	X
telephony						

Compatibility	**iPad Wi-Fi**	**iPad Wi-Fi + 3G**	**iPad 2 Wi-Fi**	**iPad 2 Wi-Fi + 3G**	**iPad Wi-Fi (3rd gen)**	**iPad Wi-Fi + Cellular (3rd gen)**
video-camera			X	X	X	X
wifi	X	X	X	X	X	X

iPod Touch Devices

Table 1-7 list the capabilities for iPod touch devices.

Table 1-7 iPod touch device compatibility

Compatibility	**iPod touch**	**iPod touch 2nd gen**	**iPod touch 3rd gen**	**iPod touch 4th gen**	**iPod touch 5th gen**	**iPod touch 5th gen 16GB (no rear-facing camera)**
accelerometer	X	X	X	X	X	X
armv6	X	X	X	X	X	X
armv7			X	X	X	X
auto-focus-camera					X	
bluetooth-le					X	X
camera-flash					X	
front-facing-camera				X	X	X
gamekit		X	X	X	X	X
gps						
gyroscope				X	X	X
healthkit					X	X
location-services	X	X	X	X	X	X

Compatibility	iPod touch	iPod touch 2nd gen	iPod touch 3rd gen	iPod touch 4th gen	iPod touch 5th gen	iPod touch 5th gen 16GB (no rear-facing camera)
magnetometer						
metal						
microphone		X	X	X	X	X
opengles-1	X	X	X	X	X	X
opengles-2			X	X	X	X
opengles-3						
peer-peer		X	X	X	X	X
sms						
still-camera				X	X	
telephony						
video-camera				X	X	
wifi	X	X	X	X	X	X

Hardware GPU Information

Since the introduction of the original iPhone, Apple has continued to improve the GPU capabilities in new iOS devices. When you write a Metal or OpenGL ES app, you need to understand the specific limits of each device you app runs on. Currently, two distinct GPU categories are in common use:

- The Apple A7 and A8 GPUs
- The PowerVR SGX 543 and 554 GPUs

Table 2-1 lists the devices that are compatible with Metal and OpenGL ES 3.0.

Table 2-1 Metal and OpenGL ES 3.0 compatible devices

Device name	GPU
iPad Air 2 Wi-Fi iPad Air 2 Wi-Fi + Cellular	Apple A8 GPU
iPad Mini 3 Wi-Fi iPad Mini 3 Wi-Fi + Cellular	Apple A7 GPU
iPhone 6 and iPhone 6 Plus	Apple A8 GPU
iPhone 5s	Apple A7 GPU
iPad Air Wi-Fi iPad Air Wi-Fi + Cellular	Apple A7 GPU
iPad Mini with Retina display Wi-Fi iPad Mini with Retina display Wi-Fi + Cellular	Apple A7 GPU

Table 2-2 lists the devices that are compatible with OpenGL ES 2.0.

Table 2-2 OpenGL ES 2.0 compatible devices

Device name	GPU
iPad Air 2 Wi-Fi iPad Air 2 Wi-Fi + Cellular	Apple A8 GPU
iPad Mini 3 Wi-Fi iPad Mini 3 Wi-Fi + Cellular	Apple A7 GPU
iPhone 6 and iPhone 6 Plus	Apple A8 GPU
iPhone 5s	Apple A7 GPU
iPhone 4s, iPhone 5, iPhone 5c	SGX 543
iPhone 3GS, iPhone 3GS (China), and iPhone 4	SGX 535
iPad Air Wi-Fi and iPad Air Wi-Fi + Cellular	Apple A7 GPU
iPad Wi-Fi (4th generation) and iPad Wi-Fi + Cellular (4th generation)	SGX 554
iPad 2 Wi-Fi, iPad 2 Wi-Fi + 3G, iPad Wi-Fi (3rd generation), and iPad Wi-Fi + Cellular (3rd generation)	SGX 543
iPad Mini with Retina display Wi-Fi and iPad Mini with Retina display Wi-Fi + Cellular	Apple A7 GPU
iPad Mini Wi-Fi and iPad Mini Wi-Fi + Cellular	SGX 543
iPad Wi-Fi and iPad Wi-Fi + 3G	SGX 535
iPod Touch (5th generation)	SGX 543
iPod Touch (3rd and 4th generations)	SGX 535

Apple A7 and A8 GPU Hardware

Together, the Apple A7 and A8 GPUs create a new generation of graphics hardware that support both Metal and OpenGL ES 3.0. To get the most out of a 3D, graphics-dominated app running on the A7 and A8 GPUs, use Metal. Metal provides extremely low-overhead access to the A7 and A8 GPUs, enabling incredibly high performance for your sophisticated graphics rendering and computational tasks. Metal eliminates many performance bottlenecks—such as costly state validation—that are found in traditional graphics APIs. If you do not want to use Metal, use OpenGL ES 3.0 when building an app. Both Metal and OpenGL ES 3.0 incorporate many new features, such as multiple render targets and transform feedback, that have not been available on mobile processors before. This means that advanced rendering techniques that have previously been available only on desktop machines, such as deferred rendering, can now be used in iOS apps. See *Metal Programming Guide* for more information about what features are visible to Metal apps.

To take advantage of the power of the A7 and A8 GPUs, your app must support Metal or OpenGL ES 3.0. Using Metal or OpenGL ES 3.0 gives you access to the new features and also to a larger pool of rendering resources. For example, on the A7 and A8 GPUs, an app that uses Metal or OpenGL ES 3.0 can access twice as many textures in a shader than an app that uses OpenGL ES 2.0.

Best Practices for OpenGL ES 3.0

These practices apply to OpenGL ES apps on Apple A7 and A8 GPU hardware:

- Avoid operations that modify OpenGL ES objects already in use by the renderer because of previously submitted drawing commands. When you need to modify OpenGL ES resources, schedule those modifications at the beginning or end of a frame. These commands include `glBufferSubData`, `glBufferData`, `glMapBuffer`, `glTexSubImage`, `glCopyTexImage`, `glCopyTexSubImage`, `glReadPixels`, `glBindFramebuffer`, `glFlush`, and `glFinish`.
- Follow the drawing guidelines found in Do Not Sort Rendered Objects Unless Necessary in *OpenGL ES Programming Guide for iOS*.
- When possible, your renderbuffer's height and width should be a multiple of 32 pixels.

OpenGL ES 3.0 on Apple A7 and A8 GPUs

Table 2-3 provides a high-level summary for OpenGL ES 3.0.

Table 2-3 OpenGL ES 3.0 attribute values implemented for the Apple A7 and A8 GPUs

OpenGL ES 3.0 attributes	Values for A7 and A8 GPUs
MAX_TEXTURE_SIZE, MAX_RENDERBUFFER_SIZE, MAX_CUBE_MAP_TEXTURE_SIZE	4096 x 4096
MAX_ARRAY_TEXTURE_LAYERS	2048
MAX_COLOR_ATTACHMENTS	4
MAX_COMBINED_FRAGMENT_UNIFORM_COMPONENTS	50048
MAX_COMBINED_TEXTURE_IMAGE_UNITS	32
MAX_COMBINED_UNIFORM_BLOCKS	24

OpenGL ES 3.0 attributes	Values for A7 and A8 GPUs
MAX_COMBINED_VERTEX_UNIFORM_COMPONENTS	51200
MAX_DRAW_BUFFERS	4
MAX_FRAGMENT_INPUT_COMPONENTS	64
MAX_FRAGMENT_UNIFORM_BLOCKS	12
MAX_FRAGMENT_UNIFORM_COMPONENTS	896
MIN_PROGRAM_TEXEL_OFFSET	-8
MAX_PROGRAM_TEXEL_OFFSET	7
MAX_SAMPLES	8
MAX_TEXTURE_IMAGE_UNITS	16
MAX_TRANSFORM_FEEDBACK_INTERLEAVED_COMPONENTS	64
MAX_TRANSFORM_FEEDBACK_SEPARATE_ATTRIBS	4
MAX_TRANSFORM_FEEDBACK_SEPARATE_COMPONENTS	4
MAX_VARYING_VECTORS	15
MAX_VERTEX_ATTRIBS	16
MAX_VERTEX_OUTPUT_COMPONENTS	64
MAX_VERTEX_TEXTURE_IMAGE_UNITS	16
MAX_VERTEX_UNIFORM_BLOCKS	12
MAX_VERTEX_UNIFORM_COMPONENTS	2048
MAX_UNIFORM_BLOCK_SIZE	16384

Considerations

The A7 and A8 GPUs process all floating-point calculations using a scalar processor, even when those values are declared in a vector. Proper use of write masks and careful definitions of your calculations can improve the performance of your shaders. For more information, see Perform Vector Calculations Lazily in *OpenGL ES Programming Guide for iOS* .

Medium- and low-precision floating-point shader values are computed identically, as 16-bit floating point values. This is a change from the PowerVR SGX hardware, which used 10-bit fixed-point format for low-precision values. If your shaders use low-precision floating point variables and you also support the PowerVR SGX hardware, you must test your shaders on both GPUs.

The Apple A7 and A8 GPUs do not penalize dependent-texture fetches.

Always use framebuffer discard operations when your framebuffer contents are no longer needed. The penalty for not doing so is higher than it was on earlier GPUs. For best results, use the `GLKView` class; it automatically implements framebuffer discard operations.

When rendering to multiple targets, limit your app to four image targets (and no more than 128 bits of total data on A7 and 256 bits of total data on A8 written to the targets). A single `sRGB` target counts as 64 bits.

Supported OpenGL ES 3.0 Extensions

OpenGL ES 3 includes functionality provided by extensions in Apple's implementation of OpenGL ES 2.0. If you are updating an existing OpenGL ES 2.0 app to use OpenGL ES 3.0, please note that many of these extensions are not provided in OpenGL ES 3.0. Therefore, your code must be updated to use the new core functionality instead.

The following extensions are supported for the A7 and A8 GPUs in OpenGL ES 3 apps:

- APPLE_clip_distance
- APPLE_color_buffer_packed_float
- APPLE_copy_texture_levels
- APPLE_rgb_422
- APPLE_texture_format_BGRA8888
- EXT_color_buffer_half_float
- EXT_debug_label
- EXT_debug_marker
- EXT_pvrtc_sRGB
- EXT_read_format_bgra
- EXT_separate_shader_objects
- EXT_shader_framebuffer_fetch
- EXT_texture_filter_anisotropic
- IMG_read_format

- IMG_texture_compression_pvrtc

The A8 GPU supports the following additional extension: GL_KHR_texture_compression_astc_ldr.

The following extensions are supported, but OpenGL ES 3 provides core functionality that matches these extensions. If you are porting an OpenGL ES 2 app that uses these extensions, you should migrate your shaders to the core OpenGL ES 3 functionality.

- EXT_shader_texture_lod
- EXT_shadow_samplers
- OES_standard_derivatives

OpenGL ES 2.0 on Apple A7 and A8 GPUs

Table 2-4 provides a high-level summary for OpenGL ES 2.0.

Table 2-4 OpenGL ES 2.0 attribute values implemented for the Apple A7 and A8 GPUs

OpenGL ES 2.0 attributes	Values for A7 and A8 GPUs
MAX_TEXTURE_SIZE, MAX_RENDERBUFFER_SIZE, MAX_CUBE_MAP_TEXTURE_SIZE	4096 x 4096
MAX_TEXTURE_IMAGE_UNITS	8
MAX_COMBINED_TEXTURE_IMAGE_UNITS	8
MAX_VERTEX_TEXTURE_IMAGE_UNITS	8
MAX_VERTEX_ATTRIBS	16
MAX_VERTEX_UNIFORM_VECTORS	128
MAX_FRAGMENT_UNIFORM_VECTORS	64
MAX_VARYING_VECTORS	8

Considerations

The A7 and A8 GPUs process all floating-point calculations using a scalar processor, even when those values are declared in a vector. Proper use of write masks and careful definitions of your calculations can improve the performance of your shaders. For more information, see Perform Vector Calculations Lazily in *OpenGL ES Programming Guide for iOS* .

Medium- and low-precision floating-point shader values are computed identically, as 16-bit floating point values. This is a change from the PowerVR SGX hardware, which used 10-bit fixed-point format for low-precision values. If your shaders use low-precision floating point variables and you also support the PowerVR SGX hardware, you must test your shaders on both GPUs.

The Apple A7 and A8 GPUs do not penalize dependent-texture fetches.

Always use framebuffer discard operations when your framebuffer contents are no longer needed. The penalty for not discarding framebuffers is higher than it was on earlier GPUs. For best results, use the `GLKView` class; it automatically implements framebuffer discard operations.

Supported OpenGL ES 2.0 Extensions

The following extensions are supported for the Apple A7 and A8 GPUs:

- APPLE_clip_distance
- APPLE_color_buffer_packed_float
- APPLE_copy_texture_levels
- APPLE_framebuffer_multisample
- APPLE_rgb_422
- APPLE_sync
- APPLE_texture_format_BGRA8888
- APPLE_texture_max_level
- APPLE_texture_packed_float
- EXT_blend_minmax
- EXT_color_buffer_half_float
- EXT_debug_label
- EXT_debug_marker
- EXT_discard_framebuffer
- EXT_draw_instanced
- EXT_instanced_arrays
- EXT_map_buffer_range
- EXT_occlusion_query_boolean
- EXT_pvrtc_sRGB
- EXT_read_format_bgra

- EXT_separate_shader_objects
- EXT_shader_texture_lod
- EXT_shader_framebuffer_fetch
- EXT_shadow_samplers
- EXT_sRGB
- EXT_texture_filter_anisotropic
- EXT_texture_rg
- EXT_texture_storage
- IMG_read_format
- IMG_texture_compression_pvrtc
- OES_depth_texture
- OES_depth24
- OES_element_index_uint
- OES_fbo_render_mipmap
- OES_mapbuffer
- OES_packed_depth_stencil
- OES_rgb8_rgba8
- OES_standard_derivatives
- OES_texture_float
- OES_texture_half_float
- OES_texture_half_float_linear
- OES_vertex_array_object

PowerVR SGX Hardware

Imagination Technologies has several useful references about PowerVR technologies:

- PowerVR SGX OpenGL ES2.0 Application Development Recommendations. Introduces the SGX Series of processors and describes how to optimize for them.
- PowerVR 3D Application Development Recommendations. Introduces 3D graphics apps and provides some "golden rules" for developing such apps.
- Imagination Technologies home page.

Best Practices for OpenGL ES 2.0

These practices apply to OpenGL ES apps on SGX Series 5 hardware:

- Avoid operations that modify OpenGL ES objects already in use by the renderer because of previously submitted drawing commands. When you need to modify OpenGL ES resources, schedule those modifications at the beginning or end of a frame. These commands include `glBufferSubData`, `glBufferData`, `glMapBuffer`, `glTexSubImage`, `glCopyTexImage`, `glCopyTexSubImage`, `glReadPixels`, `glBindFramebuffer`, `glFlush`, and `glFinish`.
- To take advantage of the processor's hidden surface removal, follow the drawing guidelines found in Do Not Sort Rendered Objects Unless Necessary in *OpenGL ES Programming Guide for iOS* .
- Vertex buffer objects (VBOs) provide a significant performance improvement on the PowerVR SGX. See Use Vertex Buffer Objects to Manage Copying Vertex Data in *OpenGL ES Programming Guide for iOS* .
- Use Core Animation rotations of renderbuffers to rotate content between landscape and portrait mode. For best performance, ensure that the renderbuffer's height and width are each a multiple of 32 pixels.

OpenGL ES 2.0 on PowerVR SGX Series 5 Hardware

Table 2-5 provides a high-level summary for OpenGL ES 2.0 platforms.

Table 2-5 OpenGL ES 2.0 attribute values implemented for SGX 543 and 554

OpenGL ES 2.0 attributes	Values for SGX 543 and 554
MAX_TEXTURE_SIZE, MAX_RENDERBUFFER_SIZE, MAX_CUBE_MAP_TEXTURE_SIZE	4096 x 4096
MAX_TEXTURE_IMAGE_UNITS	8
MAX_COMBINED_TEXTURE_IMAGE_UNITS	8
MAX_VERTEX_TEXTURE_IMAGE_UNITS	8
MAX_VERTEX_ATTRIBS	16
MAX_VERTEX_UNIFORM_VECTORS	128
MAX_FRAGMENT_UNIFORM_VECTORS	64
MAX_VARYING_VECTORS	8

Considerations

The PowerVR SGX processes high-precision floating-point calculations using a scalar processor, even when those values are declared in a vector. Proper use of write masks and careful definitions of your calculations can improve the performance of your shaders. For more information, see Perform Vector Calculations Lazily in *OpenGL ES Programming Guide for iOS* .

Although medium- and low-precision floating-point values are both processed in parallel, low-precision variables have a few specific performance limitations:

- Swizzling components of vectors declared with low precision is expensive and should be avoided.
- Many built-in functions use medium-precision inputs and outputs. If your app provides low-precision floating-point values as parameters or assigns the results to a low-precision floating-point variable, the shader may have to include additional instructions to convert the values. These additional instructions are also added when swizzling the vector results of a computation.

For best results, limit your use of low-precision variables to color values.

Supported OpenGL ES 2.0 Extensions

The following extensions are supported for all SGX Series 5 processors: 543 and 554:

- APPLE_copy_texture_levels
- APPLE_framebuffer_multisample
- APPLE_rgb_422
- APPLE_texture_format_BGRA8888
- APPLE_texture_max_level
- APPLE_sync
- EXT_blend_minmax
- EXT_debug_label
- EXT_debug_marker
- EXT_discard_framebuffer
- EXT_draw_instanced
- EXT_instanced_arrays
- EXT_map_buffer_range
- EXT_separate_shader_objects
- EXT_shader_framebuffer_fetch
- EXT_read_format_bgra

- EXT_shader_texture_lod
- EXT_texture_filter_anisotropic
- EXT_texture_storage
- IMG_read_format
- IMG_texture_compression_pvrtc
- OES_depth24
- OES_depth_texture
- OES_element_index_uint
- OES_fbo_render_mipmap
- OES_mapbuffer
- OES_packed_depth_stencil
- OES_rgb8_rgba8
- OES_standard_derivatives
- OES_texture_half_float
- OES_texture_float
- OES_texture_half_float
- OES_vertex_array_object

The following extensions are supported for the SGX 543 and 554 processors only:

- EXT_color_buffer_half_float
- EXT_occlusion_query_boolean
- EXT_pvrtc_sRGB
- EXT_shadow_samplers
- EXT_sRGB
- EXT_texture_rg
- OES_texture_half_float_linear

Document Revision History

This table describes the changes to *iOS Device Compatibility Reference* .

Date	Notes
2015-01-12	Updated OpenGL ES extension list with an A8 specific extension.
2014-10-24	Changed the GPU associated with iPad Mini 3's to the A7 chip.
2014-10-20	Updated to include new iPad information.
2014-09-17	Added new device compatibility information.
2014-02-11	Added opengles-3 compatibility key.
2013-10-22	Added new hardware devices. Added links to OpenGL ES extensions introduced in iOS 7.
2013-09-18	New document that describes details about the features of existing iOS devices.