The Flutter Engine will migrate to using the Metal Rendering API instead of OpenGL on iOS.

Why use Metal?

A Flutter user interface is displayed on screen by accessing the GPU on the platform using an accelerated graphics API. This API has been some version of OpenGL ES on iOS. OpenGL ES and associated APIs were formally deprecated by Apple in 2018 with the introduction of iOS 12. Metal is now the recommended API for low-level, low-overhead rendering on iOS and Mac. While the cross platform aspects of OpenGL were certainly useful, the Flutter team agrees with the case for Metal. The deprecation of OpenGL ES also appears to have resulted in a less well maintained OpenGL driver on iOS. There is an uptick in iOS specific OpenGL issues that are increasingly harder to debug and unlikely to be fixed. For these reasons, a migration to Metal on iOS is imperative.

Where will Flutter use Metal?

Metal will be used on Apple A7 devices or later running iOS 10 or above. When Metal cannot be used, OpenGL will be used instead.

Why can't Flutter always use Metal on iOS?

Flutter currently supports running on iOS 8 and above. This includes devices on which Metal itself is not available. Specifically, Metal is only available on iOS devices with Apple A7 or later. This would exclude devices like the iPhone 5C that can be updated to iOS 10 but only have an Apple A6 SOC. A policy that mandates the use of Metal everywhere would mean we would have to leave some devices behind. That is not something we are prepared to consider now.

That still leaves a small number of devices that support iOS versions between 8 and 10 that have an Apple A7 SOC (or later) but have not updated to the latest iOS versions. Flutter could technically use Metal on these devices as well but currently doesn't. This is because versions of the Metal API on earlier versions of iOS weren't expressive enough for the features needed by Flutter (specifically the Skia rendering library used by Flutter). This requires authoring workarounds or fallbacks for cases like dual-source blending. To reduce the implementation and testing burden it was decided to pick iOS 10 as the baseline instead. This decision can be revised if necessary.

Does this change the minimum version of iOS supported by Flutter?

No. Flutter will use Metal on newer devices and iOS versions but the OpenGL backend remains as-is and present. It will be used when Metal is unavailable.

As a Flutter Developer, is there anything I need to do to migrate to using Metal on iOS?

Except for the highly unlikely case that your application makes assumptions about Flutter using OpenGL under the hood, **no**. The updated Flutter Engine requires no changes to the Flutter application or how it is embedded.

Is there an impact to the binary size because of the addition of Metal support?

Yes. The size of the uncompressed Flutter Engine binary will increase by ~250KB. Incremental reductions to this size will continue to be made but a significant reduction requires the removal of the OpenGL backend. This is not something we have considered yet.

Will plugins for the camera preview, video playback, webview, maps, etc. keep working?

Yes. There are no updates necessary to these plugins and they will continue to work as expected. All plugins in the Flutter Plugins repo have also been tested to work with the Metal backend.

How can I tell if my application makes assumptions about Flutter using OpenGL under the hood?

The rendering API used by Flutter is an implementation detail of the Flutter Engine. However, Flutter does provide the ability for the underlying platform to provide textures for inline composition within a Flutter hierarchy. Developers (or plugins) do this by using a Texture widget in their applications and writing some platform specific code to provide and update these textures. If the platform specific code that operates on the textures provided to the Flutter Engine does not specify that the texture may interoperate with the Metal API, inline composition of these textures may fail. For guidance on ensuring your textures are interoperable with Metal, please follow the documentation on how to "Create an Interoperable Texture". Specifically, the `kCVPixelBufferMetalCompatibilityKey` must be specified in addition to the `kCVPixelBufferOpenGLCompatibilityKey` key when creating the pixel buffer describing the texture is given to Flutter.

If you application does not use the Texture widget, there is nothing to do. If it does, the texture provided to Flutter must be checked to see that they are interoperable with Metal.

This is a very unique use case and no plugins in the Flutter Plugins repo use this mechanism.

If my application uses OpenGL to provide textures for inline composition in a Flutter application, will this work with the Metal backend?

Yes. As mentioned earlier, as long as the textures provided to Flutter are interoperable with Metal, you may continue to use OpenGL to update the texture and Flutter will composite the same using Metal.

How can I tell at runtime if my Flutter application is using Metal to render instead of OpenGL?

The `FlutterViewController`s view will have a class that is a `CAMetalLayer`. Flutter will always use either Metal or OpenGL. That is, in a process, there is no chance of some Flutter application instances using Metal and others OpenGL.

Will Flutter use Metal on iOS Simulators?

No. Not currently. Flutter will continue to use the software backend on iOS simulators as before. Flutter on simulators is something on the radar though.