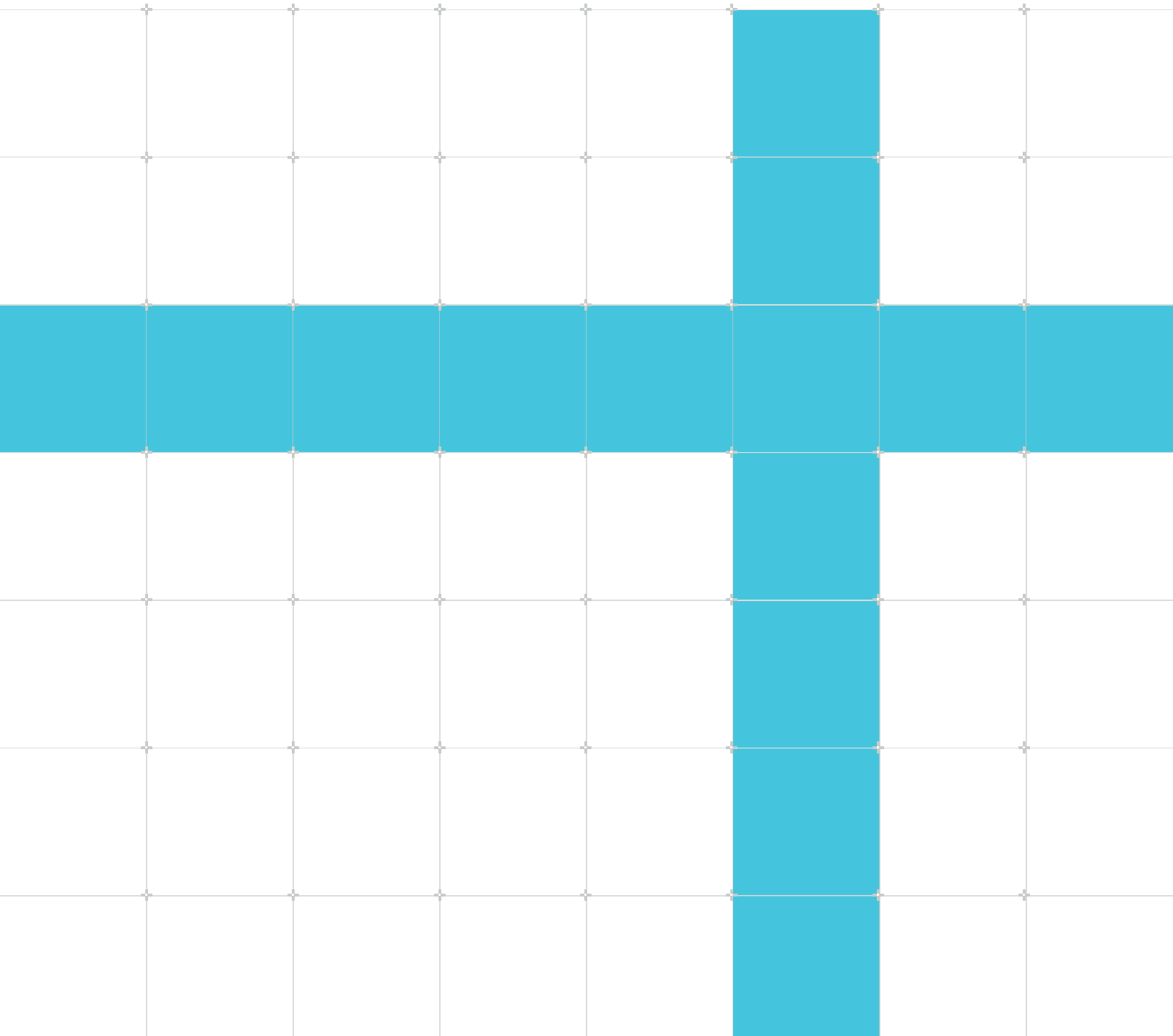
Developing games with Vulkan in Unity

Version 1.0

Guide

Issue 01
102339_0100_01_en



Developing games with Vulkan in Unity
Guide

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	17 December 2020	Non-Confidential	First release

Proprietary Notice

and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Overview

Unity is a games development platform that can use different graphics and compute APIs to render mobile games. One of these APIs is Vulkan, from the Khronos group that created OpenGL. This guide reviews the benefits of Vulkan by following a case study of the mobile game Sky Force Reloaded from Infinite Dreams.

The Vulkan driver is simpler and more efficient than OpenGL, reducing GPU and power consumption. The simplicity means the low-level access work moves to the application, which can make the application development work more complex. To keep the application development work simple, Unity handles the low-level access on behalf of the developer. Game developers can therefore enjoy the advantages of Vulkan without adding work for themselves.

The Sky Force Reloaded development team moved from OpenGL ES to Vulkan to improve the graphics of the game, while lowering its power consumption. The team used Unity to work with both APIs, so that they did not need to redevelop the game when switching from OpenGL ES to Vulkan.

By the end of this guide, you will have insight into how Vulkan can help the performance of your own game, and how to change the API that your game uses.

2. Using Vulkan in Unity

Implementing Vulkan can be a complex job, but Unity handles the complexity on behalf of game developers. This means that switching from OpenGL or OpenGL ES to Vulkan does not require any development work. Unity implements the change on its own.

Differences between OpenGL ES and Vulkan

There are many differences between Vulkan and the OpenGL and OpenGL ES APIs. These differences include Vulkan:

- Providing a unified API for mobile, desktop, console, and embedded systems, and being portable across a wide range of implementations.
- Using a simpler driver to minimize overhead and reduce application processor bottlenecks. The driver has lower latency and is more efficient than OpenGL, so the application achieves better performance. The application, instead of the driver, manages resources and has direct, low-level control of the GPU.
- Supporting multithreading across multiple application processors. The game can therefore use multiple application processors efficiently, lowering processing load and power consumption. The application itself manages threads and synchronization.
- Using command buffers instead of direct function calls to execute commands. You can use multithreading to create parallel buffers, and submit buffers to different device queues, for example graphics, compute, and DMA. Having separate queues provides flexibility for job creation. Multithreading runs the game on multiple application processor cores, which improves performance. Also, using multiple processors at a lower clock rate, instead of a faster clock rate single processor, reduces power consumption.
- Using SPIR-V, a multi-API, intermediate language for parallel compute and graphics. Using SPIR-V means that there is no front-end compiler in the Vulkan driver. This makes the driver simpler, and the shader compilation faster. You can use the same SPIR-V front-end compiler on multiple platforms to generate pre-compiled shaders. With SPIR-V, you do not have to ship shader source code with your application. You also have the option to use different shading languages in the future.
- Loading validation layers in your development environment for testing and debugging, and shipping to production without those layers.
- Performing multi-pass rendering, where each sub-pass in a group provides a different output. Vulkan optimizes the order of grouped sub-passes so that each sub-pass can access information provided by the previous sub-passes. This helps Vulkan reduce memory use and hold data on the fast on-chip memory, which saves bandwidth and power.


Note

Mutli-passing is more efficient on tile-based GPUs like Mali GPUs.

Changing the API in Unity to Vulkan

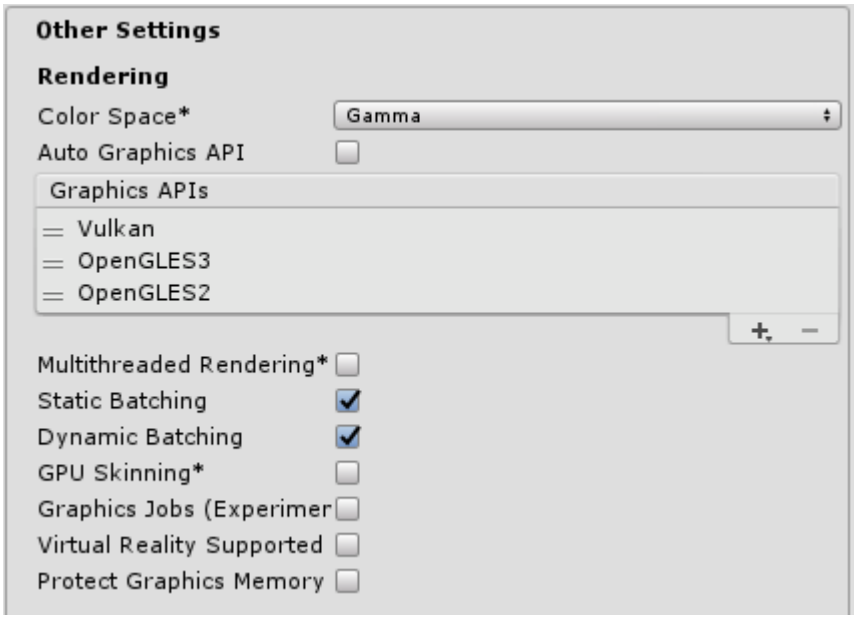
Unity 2019.3 and newer use Vulkan by default. If you are on an older version, or want to change APIs for an existing game, you will need to manually add it to the API list and make it your new default API.

To add the Vulkan API and make it default:

1. Select File > Build Settings...
2. Click Player Settings...
3. Uncheck Auto Graphics API in the Other Settings panel to allow manual API selection.
4. Click the plus button and select Vulkan from the list. Vulkan is added as the last option on the list.
5. To use Vulkan, either:

- Move it to the top of the API list, as shown in this image:

Figure 2-1: Selecting Vulkan as the default graphics API



- Remove all other APIs from the list by selecting an API and clicking the minus button.

3. Sky Force Reloaded case study

Sky Force Reloaded from mobile game developer Infinite Dreams is an action game that is set in a rich graphics environment. The game demands a lot from both the application processor and the GPU. Infinite Dreams first developed the game with OpenGL ES in Unity, and then started examining optimization options.

Choosing what to optimize

Sky Force Reloaded is a very rich game, displaying many objects at the same time. The team decided that the area likely to yield the best optimization would be fill rate.

Fill rate is the speed at which the GPU can draw frames every second. Gamers like to see 60 frames per second (FPS), but performance issues can make the FPS drop. Usually, you can eliminate fill rate problems by decreasing the resolution of the frame buffer. But Sky Force Reloaded had performance problems even at a low resolution and on high-end devices, and could not always maintain 60 FPS.

The team learned that the game was making as many as 1,000 draw calls per frame. To prepare data for the GPU for those draw calls, the OpenGL ES driver was keeping the application processor busy for long periods. Even on the high-end devices, this much work on the application processor was causing the device to slow down. In other words, every draw call has a computation overhead, so the large number of calls the game makes is computationally expensive.

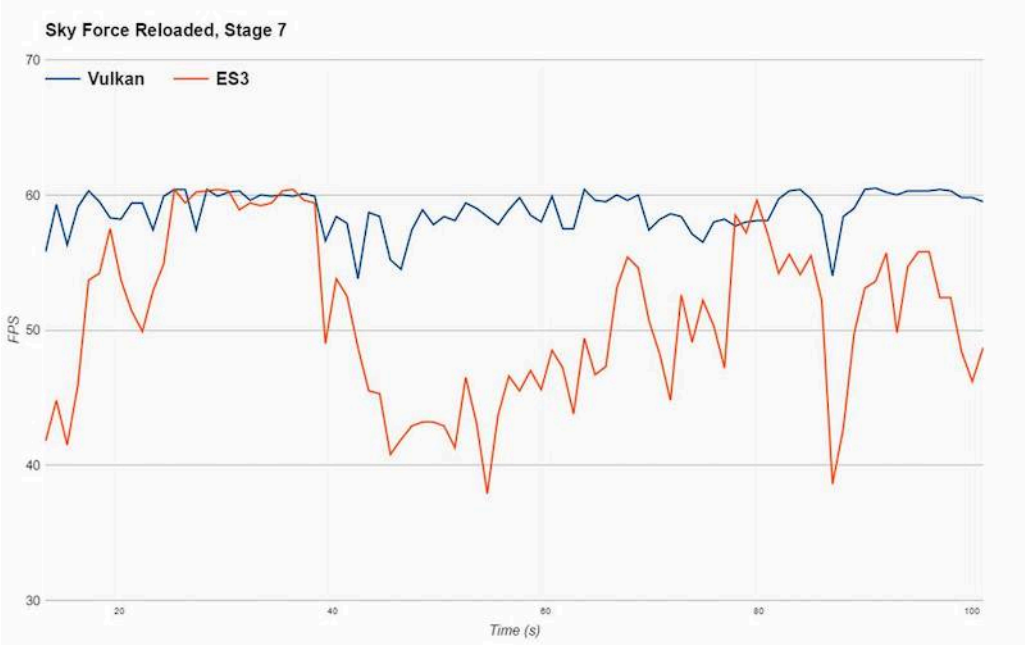
With OpenGL ES, the team had two options to optimize the fill rate: minimize the number of draw calls, or modify the calls so that the game engine could batch them. However, both options can reduce the quality of the game. So the team decided to see what sort of optimization Vulkan can provide.

Testing OpenGL ES and Vulkan on Sky Force Reloaded

The team decided to compare the performance of the game using Vulkan and OpenGL ES. They created a benchmark from the slowest part of the game, where OpenGL ES could not deliver 60 FPS. Using a specific scene of the game as a benchmark meant that the team could compare both APIs directly.

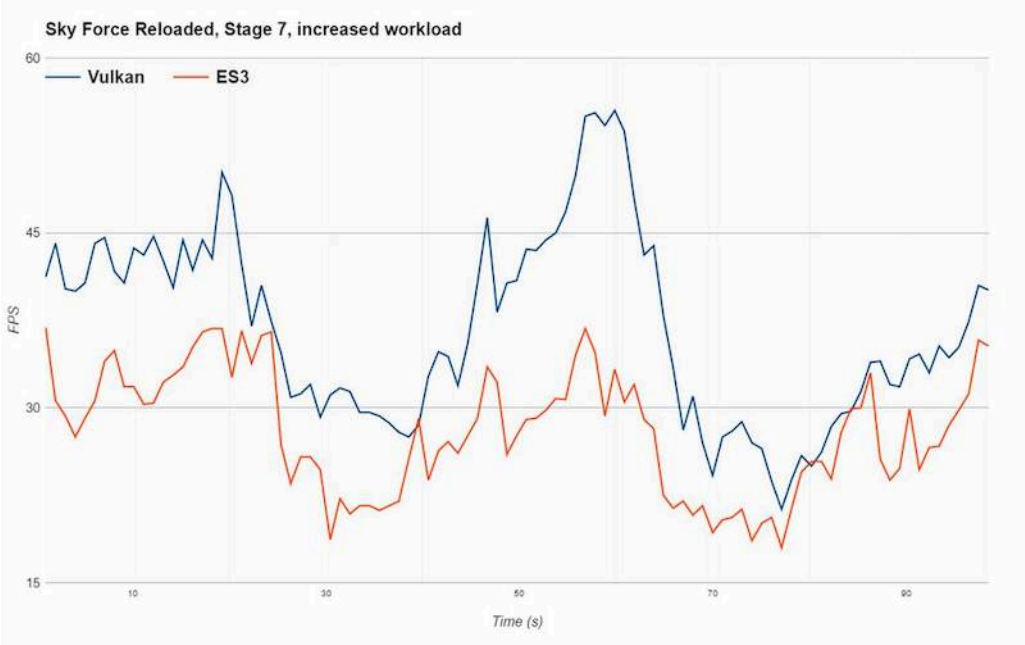
Performance was measured by the total time, in seconds, that each API could provide 60 FPS. In the following graph, you can see that the Vulkan performance is a 15% improvement on the OpenGL ES performance:

Figure 3-1: Comparing OpenGL ES and Vulkan on the same scene



The team decided to push the APIs a bit further. They added more objects to the benchmark scene, until even Vulkan was struggling to provide 60 FPS. With all the additional objects, the performance gap between the APIs grew. Vulkan was now 32% more efficient than OpenGL ES, as measured as total time each API could provide 60 FPS. The graph shows the FPS for each API as they rendered the benchmark scene:

Figure 3-2: Comparing OpenGL ES and Vulkan with more objects in the scene

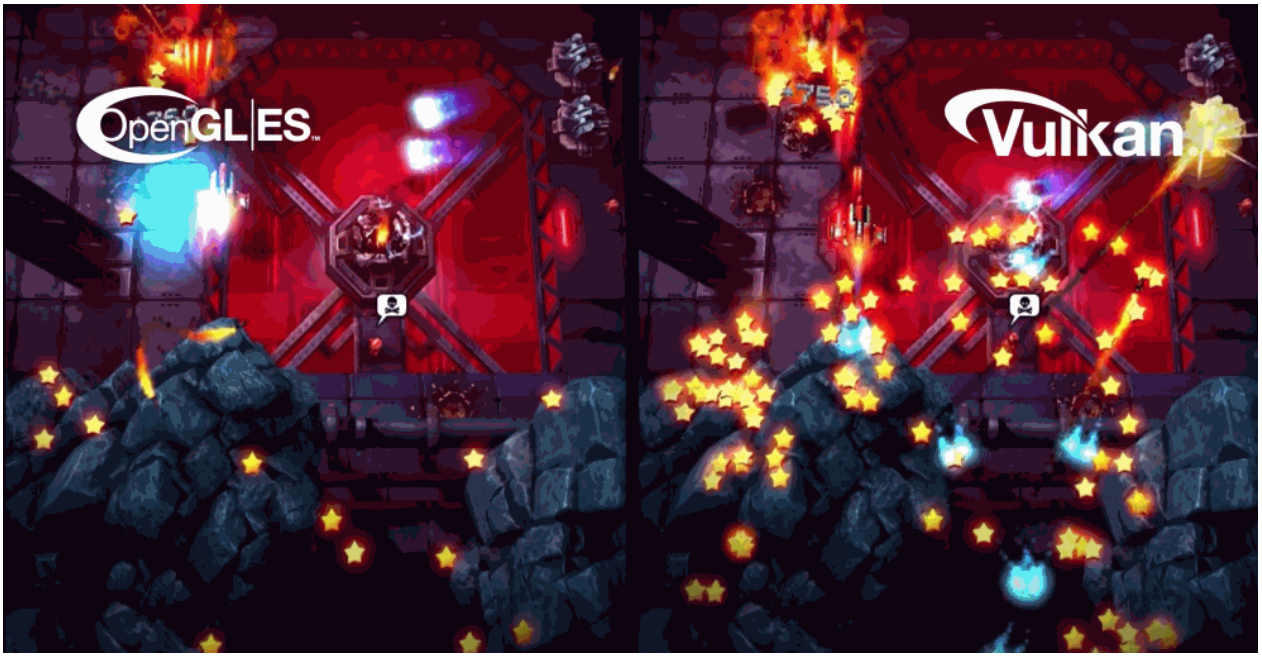


The test clearly showed that when using Vulkan, the team could add more objects, animations, and particles without sacrificing their FPS. The game looked even richer, without requiring changes other than the API switch.

Comparing object quantities at 60 FPS

Next, the team checked how many objects Vulkan can render without negatively impacting the FPS. The team found that at a frame rate of 60 FPS, Vulkan can render six times more stars and twice as many bullets as OpenGL ES. Our side-by-side video demonstrates the difference, as shown in this screen capture:

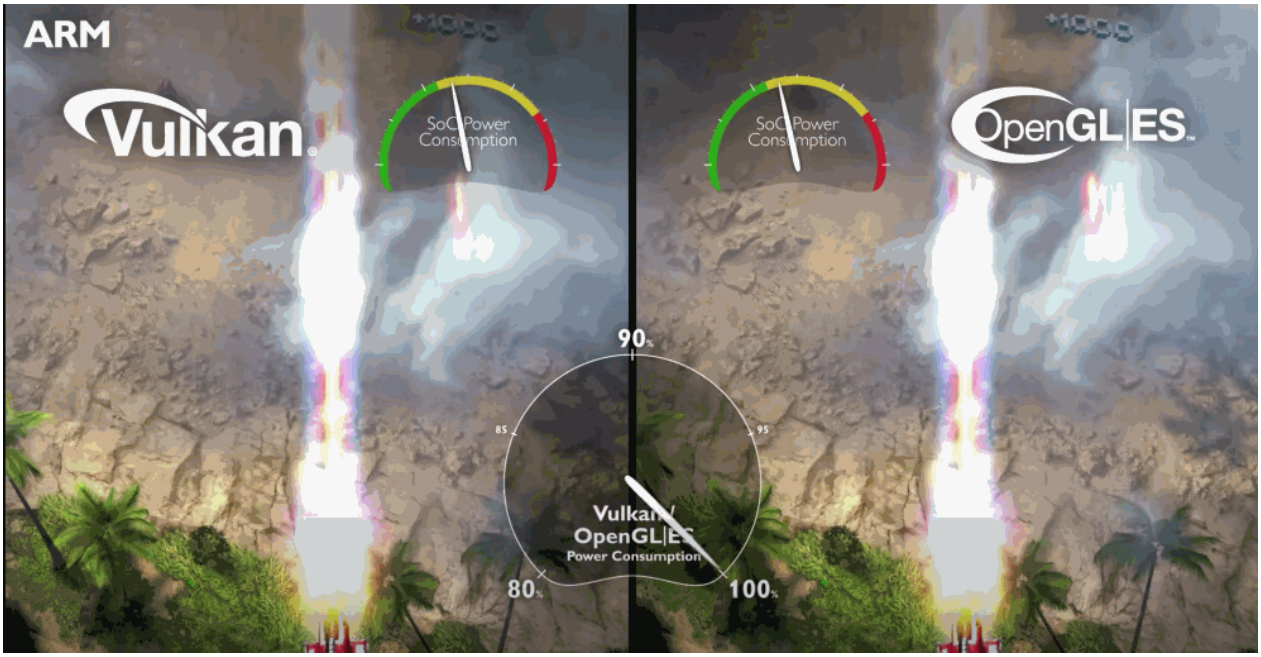
Figure 3-3: Side-by-side comparison of OpenGL ES and Vulkan object quantities



Comparing power consumption

Sky Force Reloaded is a game that is GPU and processor intensive, leading some players to complain of battery drain. The team did not want to compromise on the console-like graphics, so they compared power consumption with Vulkan and OpenGL ES. Vulkan lowered power consumption by 10-12%, increasing play time for a given battery charge. You can see a side-by-side comparison on YouTube, as shown in this screen capture:

Figure 3-4: Side-by-side comparison of Vulkan and OpenGL ES power consumption



Case study results

By switching Sky Force Reloaded to Vulkan, the development team gained better graphics at a lower power consumption. Because they relied on Unity to interface with Vulkan, the switch did not require development work.

4. Related information

Here are some resources related to material in this guide:

- Arm community - Ask development questions, and find articles and blogs on specific topics from Arm experts.
- Arm Developer: Graphics and Gaming Development
- Arm Guide for Unity Developers
- Sky Force Reloaded on the Infinite Dreams site

5. Next steps

In this guide, we saw how switching the Unity API from OpenGL ES to Vulkan helped the game development team get better performance, without adding development work. You can try Vulkan on your own games and see what results you get.

Further guides in this series introduce other aspects of game development in Unity. To keep learning, see more in our series of guides.