Vulkan Development for Apple Environments

Richard Wright
LunarG, Inc.

Presented at the Khronos Vulkanised 2023 Conference

LUNARG

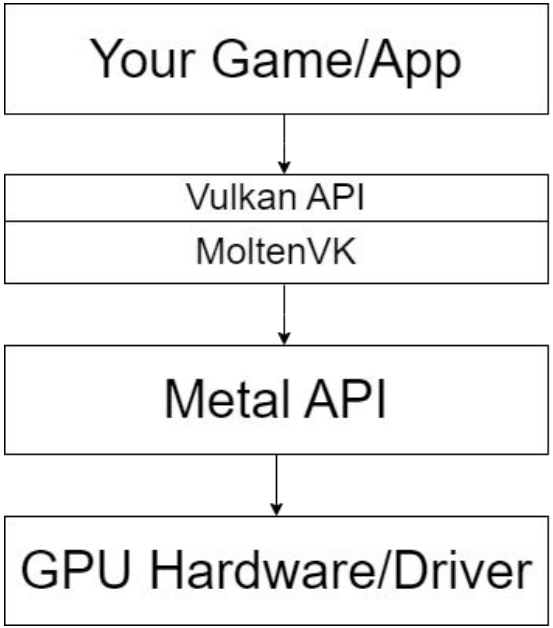
Overview

- No native Vulkan “driver” on macOS?
- How MoltenVK provides a layered approach to making a Vulkan ICD
- Shipping a “Vulkan” application on macOS and iOS
- Validation Layers and the Vulkan Configurator
- How to use the Vulkan Portability Enumeration Extension
- How to use the Portability Subset Extension

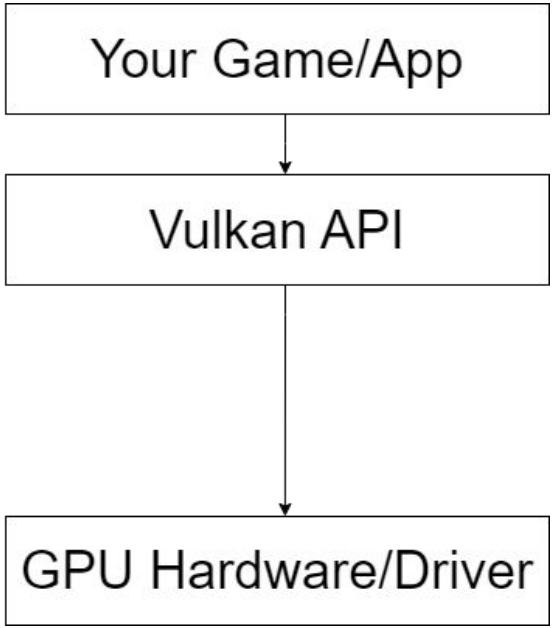
Apple does things its own way

- A bastion of openness, Apple is not
- Apple worked with IHVs (AMD/NVIDIA/Intel) to produce the low-level drivers for GPU hardware (except for Apple Silicon of course)
- The developer-facing API is Metal, a proprietary Apple-only API
- Metal is a low-level, explicit, and thin API… much like Vulkan in some ways
- Simple solution: Write a Vulkan ICD on top of Metal
- Tada - MoltenVK!
- You do not have to learn Metal, you do not have to learn two APIs. MoltenVK is just Vulkan

Vulkan/MoltenVK Layered Approach



Native Vulkan Drivers



*It's that simple…

Where do you get this magic library?

It is included in the Vulkan SDK available free at:
vulkan.lunarg.com

OR

https://github.com/KhronosGroup/MoltenVK
If you like building things yourself

The Many Faces of MoltenVK

- **System Wide Loader/ICD**
  - Useful for development
  - Works seamlessly with the vkconfig and the validation layers
  - **DO NOT SHIP** your applications expecting this

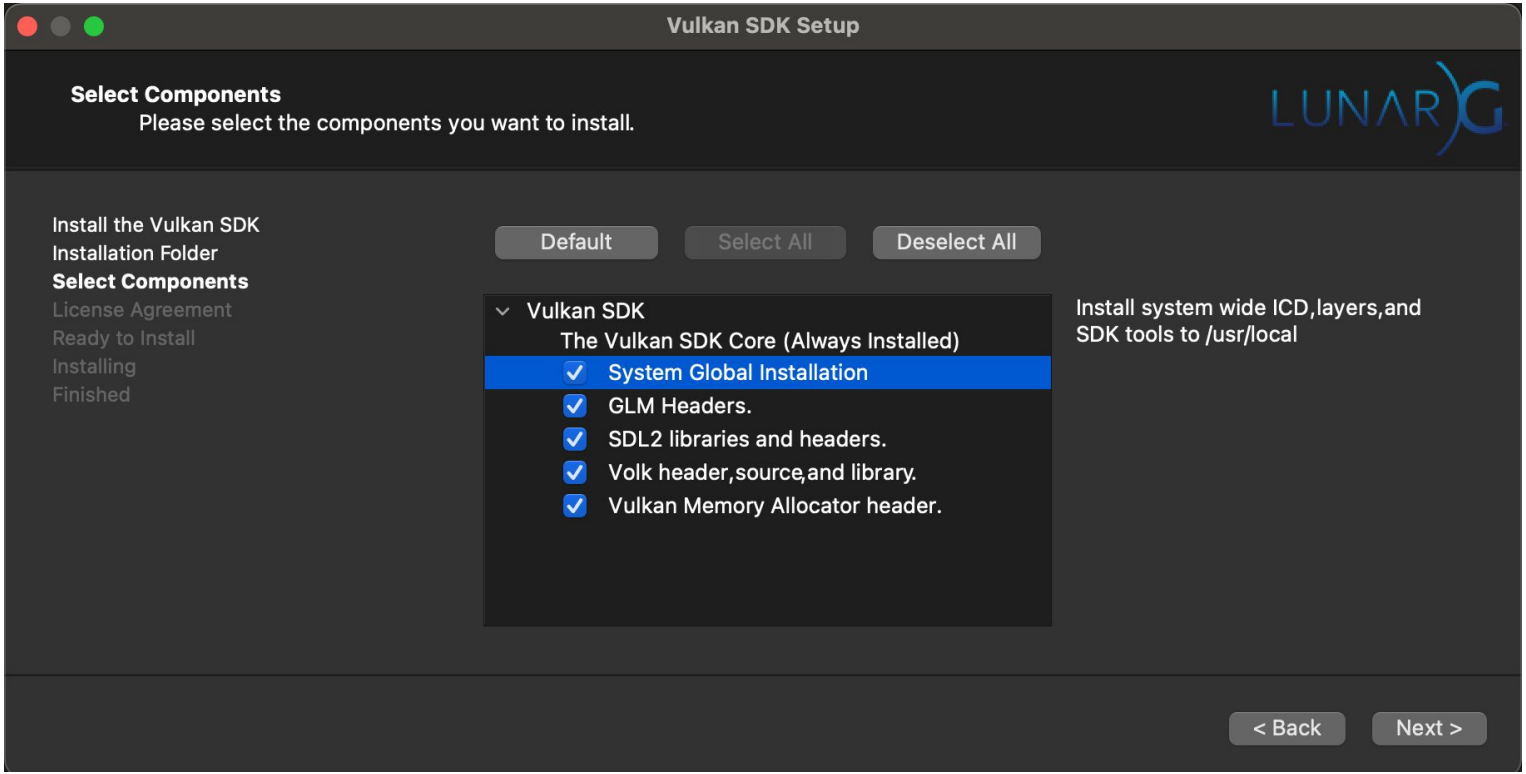
- **Include loader/MoltenVK in your app bundle**
  - Works with the loader, vkconfig, and validation layers

- **Link dynamically, embed in your bundle (in /Frameworks)***
  - Does not work with the loader, vkconfig, or validation layers

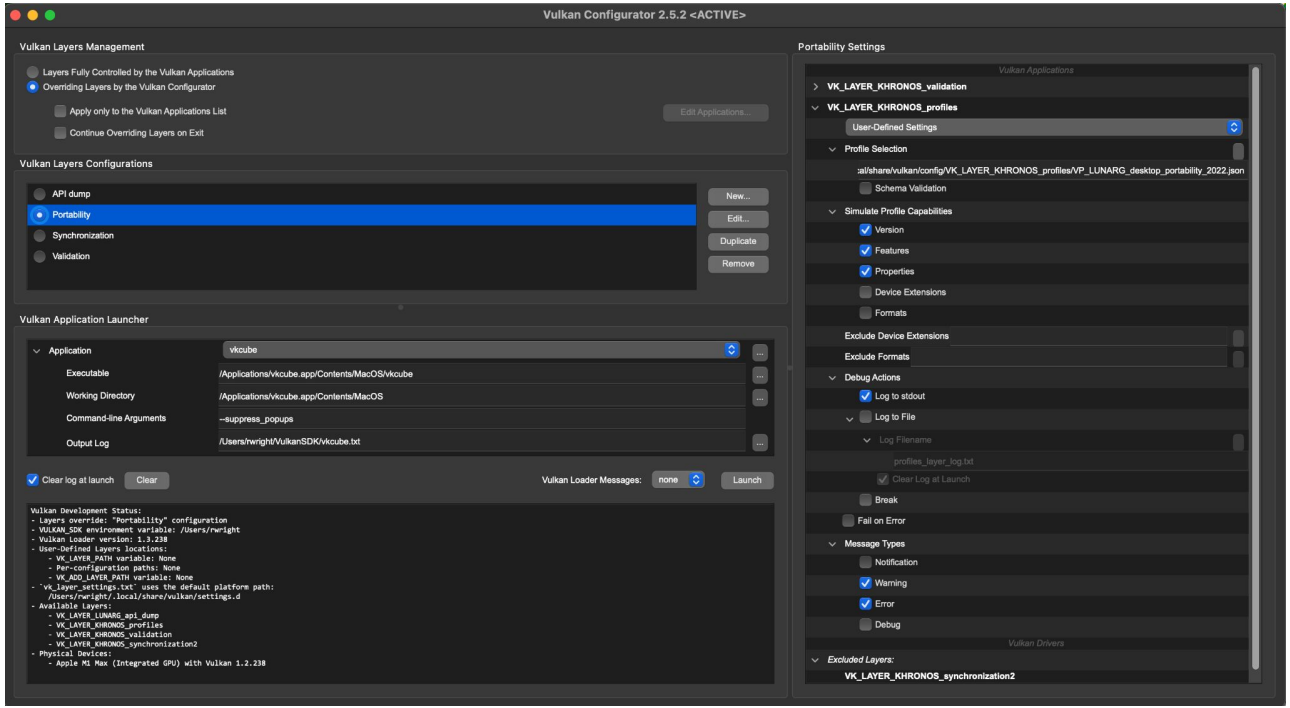
- **Link statically***
  - Does not work with loader, vkconfig, or validation layers
  - Does allow for non bundled executables to use Vulkan

*Must use one of these for iOS or tvOS

System Wide Loader/ICD



Vulkan Configurator “Just Works”

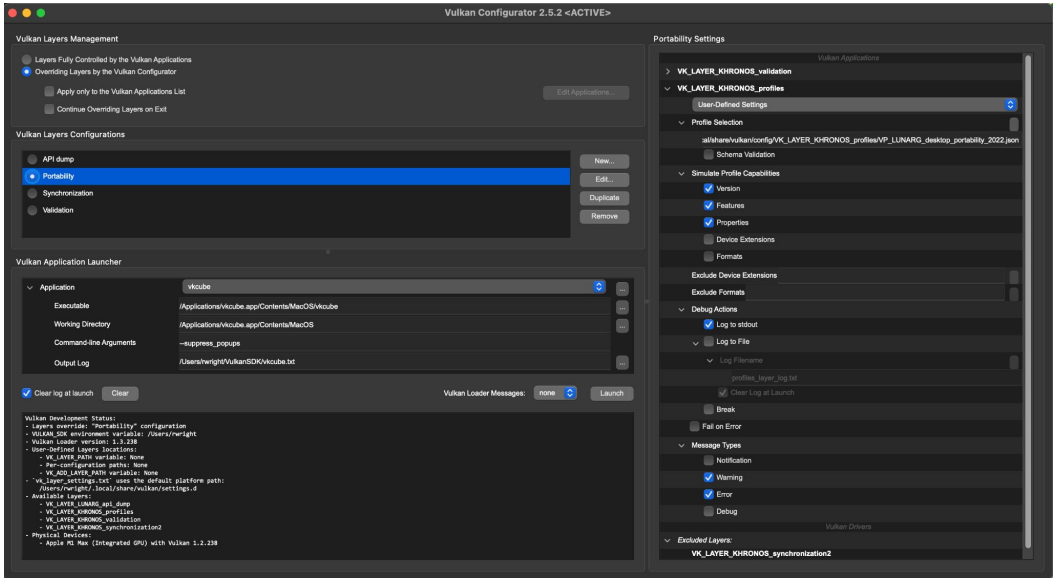


Vulkan Configurator “Just Works”

Bugs you know about

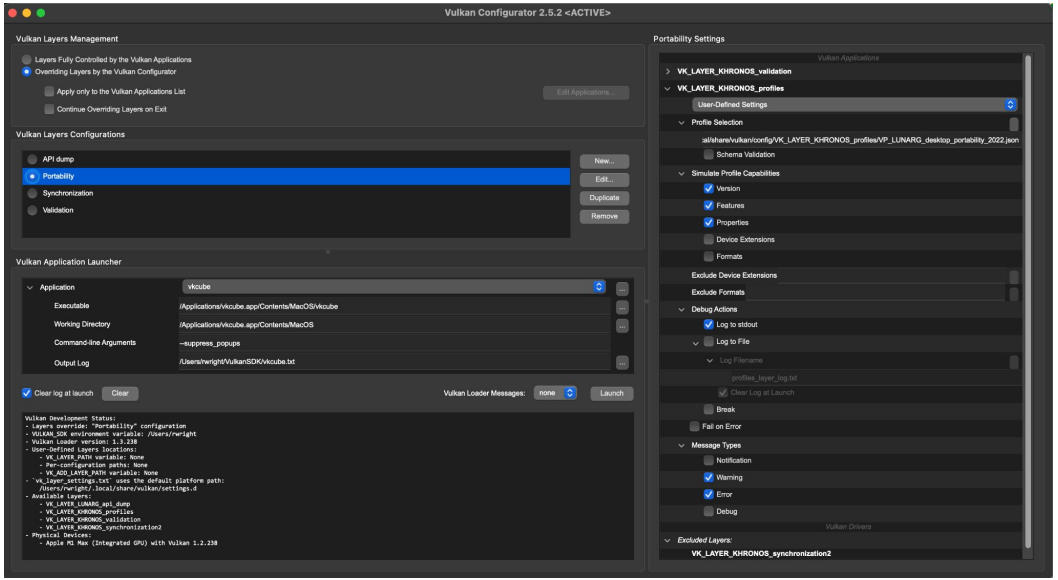
Bugs you DON’T know about

-API Usage Bugs-



Vulkan Layers on macOS

- Khronos Validation Layer
  - No DebugPrintf
  - No GPU/AV
- Khronos synchronization2
- Khronos profiles
- API Dump



Bundled Loader (macOS only)

```
VulkanRocks.app
    /Contents
        /Frameworks
            libMoltenVK.dylib
            libvulkan.1.[version number].dylib
            libvulkan.1.dylib -> libvulkan.1.[version number].dylib
        /MacOS
            VulkanRocks
        /Resources
            /vulkan
                /icd.d
                    MoltenVK_icd.json
```

https://vulkan.lunarg.com/doc/sdk/latest/mac/getting_started.html

Include a Dynamic Library

- MoltenVK is a dynamic library and can be placed in /Frameworks in the app bundle
- Simple, easy to replace. Just like any other dynamic library you might use
- Works on all Apple Platforms
- This bypasses the loader - no validation layers!
- MoltenVK has all the loader entry points, so it can “fake” the loader, but it doesn’t actually load layers, etc.

Static Link

- MoltenVK can also be linked to your app as a static library.
- Include the MoltenVK.xcframework
- This contains static libraries for each platform
  - macOS
  - iOS/Simulator
  - tvOS/Simulator
- Great option for shipping applications - especially non-bundled apps
  - Works on all Apple devices.
  - Cannot use any layers (validation or otherwise)

Okay, that's the overview of linking and packaging…

What about the code?

There are two important extensions you need to know about if you are going to target Apple devices… in fact, this goes for ANY layered Vulkan implementation on ANY platform.

`VK_KHR_portability_enumeration`

`VK_KHR_portability_subset`

Portability Enumeration Extension

Provisional - September 2021

The purpose of this extension is to keep games/apps from “accidentally” selecting an incomplete (but Portability conformant) Vulkan Implementation*. While important today on macOS, it may be more important soon on Windows and Linux.

*This does require that a layered, Portability Conformant Vulkan implementation must identify itself to be so by supporting this extension.

Portability Enumeration Extension

This is an instance extension. You are telling the Loader what devices you want to see.

1. If “VK_KHR_portability_enumeration” is listed by vkEnumerateInstanceExtensionProperties, it means you have a (newish) loader that supports the Vulkan Portability Extension. You must add the extension name to the ppEnableExtensions list in the VkInstanceCreateInfo structure if you want to make use of a portability implementation.
2. You must also add the VK_INSTANCE_CREATE_ENUMERATE_PORTABILITY_BIT_KHR flag to the flags member.

If you do not do BOTH of the above (on macOS currently), you will get VK_ERROR_INCOMPATIBLE_DRIVER from vkCreateInstance

Portability Enumeration Extension

Important: If multiple drivers are found, and one is “portable,” and you’ve not enabled this extension, you will only see the conformant hardware driver.

This will likely happen on Windows/Linux before it happens on macOS!

Portability Enumeration Extension

```
////////////////////////////////////////////////////////////////////////
// Get the list of instance extensions
uint32_t extensionCount = 0;
vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount, nullptr);


std::vector<VkExtensionProperties> extensions(extensionCount);
vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount, extensions.data());
```

Look for the ones you want

```
std::vector<const char *> extNames;
bool bPortableEnumeration = false;
for (uint32_t i = 0; i < extensionCount; i++) {

    // If the extension is present, you must use it to get portable implementations
    if(!strcmp(extensions[i].extensionName, VK_KHR_PORTABILITY_ENUMERATION_EXTENSION_NAME))
        {
        bPortableEnumeration = true;
        extNames.push_back(VK_KHR_PORTABILITY_ENUMERATION_EXTENSION_NAME);
        }

    ...
    ...
    }
```

Create the Vulkan Loader Instance

```
VkInstanceCreateInfo inst_info = {};
inst_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
inst_info.pNext = NULL;
inst_info.pApplicationInfo = &appInfo;
inst_info.enabledLayerCount = 0;
inst_info.ppEnabledLayerNames = nullptr;
inst_info.enabledExtensionCount = (int)extNames.size();
inst_info.ppEnabledExtensionNames = extNames.data();

if(bPortableEnumeration)
    inst_info.flags |= VK_INSTANCE_CREATE_ENUMERATE_PORTABILITY_BIT_KHR;

// Create the Instance
lastResult = vkCreateInstance(&inst_info, NULL, &vulkanInstance);
```

Create the Vulkan Loader Instance

```
// Create the Instance
lastResult = vkCreateInstance(&inst_info, NULL, &vulkanInstance);
```

Forget one of these two things? With SDK/Loader 1.3.216 or later, you will get the dreaded:

```
lastResult == VK_ERROR_INCOMPATIBLE_DRIVER
```

LUNARG

So, now you've told the loader you are interested in a "Portability conformant" driver. You got one.

Now what?

Portability Subset Extension

A layered implementation of Vulkan may have some gaps in it's capabilities. This extension gives you the ability to query for missing features so you can work around them, or simply punt and tell the user you cannot run using this hardware device.

Version (provisional) 1.0 of this extension lists a specific set of features that may or may not be present… we'll get to those soon.

Portability Subset Extension

This is a **device** extension.

vkEnumerateDeviceExtensionProperties will list “VK_KHR_portability_subset”

Yep, add it to the ppEnabledExtensionNames member of VkDeviceCreateInfo.

Portability Subset Extension

```
// We have a physical device, now we need a list of it's extensions
uint32_t deviceExtensionCount;
vkEnumerateDeviceExtensionProperties(physicalDevice, nullptr, &deviceExtensionCount, nullptr);
std::vector<VkExtensionProperties> deviceExtensions(deviceExtensionCount);
vkEnumerateDeviceExtensionProperties(physicalDevice, nullptr, &deviceExtensionCount,
                                                                deviceExtensions.data());
std::vector<const char *> extNamesDevice;

for (uint32_t i = 0; i < deviceExtensionCount; i++){
    if(strcmp(deviceExtensions[i].extensionName, "VK_KHR_portability_subset") ==
0)
            extNamesDevice.push_back(deviceExtensions[i].extensionName)
    }
```

Portability Subset Extension

Query for what features are available/missing

```
VkPhysicalDevicePortabilitySubsetFeaturesKHR portabilityFeatures = {};

portabilityFeatures.sType =
            VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PORTABILITY_SUBSET_FEATURES_KHR
;

VkPhysicalDeviceFeatures2 physicalDeviceFeatures2 = {};
physicalDeviceFeatures2.sType =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FEATURES_2;
physicalDeviceFeatures2.pNext = & portabilityFeatures;
vkGetPhysicalDeviceFeatures2(physicalDevice, &physicalDeviceFeatures2);
```

Note vkGetPhysicalDeviceFeatures2 is an extension prior to Vulkan 1.1

The structure is basically a set of flags…

```
typedef struct VkPhysicalDevicePortabilitySubsetFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           constantAlphaColorBlendFactors;
// 1
    VkBool32           events;                                  // 1
    VkBool32           imageViewFormatReinterpretation;         // 0
    VkBool32           imageViewFormatSwizzle;                  // 1
    VkBool32           imageView2DOn3DImage;                    // 1
    VkBool32           multisampleArrayImage;                   // 1
    VkBool32           mutableComparisonSamplers;
// 1
    VkBool32           pointPolygons;                           // 0
    VkBool32           samplerMipLodBias;                       // 0
    VkBool32           separateStencilMaskRef;                  // 1
    VkBool32           shaderSampleRateInterpolationFunctions;  // 1
    VkBool32           tessellationIsolines;                    // 0
    VkBool32           tessellationPointMode;                   // 0
    VkBool32           triangleFans;                            // 0
    VkBool32           vertexAttributeAccessBeyondStride;       // 1
} VkPhysicalDevicePortabilitySubsetFeaturesKHR;
```

Latest values on an M1 Mac (might be different on other Mac's/GPU's)

Zero means the feature is not present on this device

THESE ARE "SUBJECT" TO CHANGE!!

AS IN "LIKELY"...

You must enable the ones you want!

```
VkDeviceCreateInfo createInfo = {};
createInfo.sType = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
createInfo.pQueueCreateInfos = &queueCreateInfo;
createInfo.queueCreateInfoCount = 1;
createInfo.pEnabledFeatures = loader.getPhysicalDeviceFeatures(nDeviceIndex);
createInfo.enabledExtensionCount = (int)extNamesDevice.size();
createInfo.ppEnabledExtensionNames = extNamesDevice.data();

createInfo.pNext =
        (VkPhysicalDevicePortabilitySubsetFeaturesKHR*)&portabilityFeatures;


logicalDevice = VK_NULL_HANDLE;
VkResult result = vkCreateDevice(physicalDevice, &createInfo, nullptr, &logicalDevice);

if (result != VK_SUCCESS)
    return false;
```

Conclusion

- MoltenVK is just a “Layered Vulkan Implementation”
- Work around missing extensions and features like any other platform
- Portability extensions (two of them) are there to help navigate this
- Performance is very good
- Try it, you’ll like it!
- Be sure and catch Bill Hollings talk: “MoltenVK: Application portability with Vulkan on Metal”

Help Us Improve the Vulkan SDK and Ecosystem

Share Your Feedback

Take the LunarG annual developer's survey

- Survey results are tabulated
- Shared with the Vulkan Working Group
- Actions are assigned
- Results are reported

Survey closes February 27, 2023

https://www.surveymonkey.com/r/PVM92RH

LUNARG30

Resources

This Presentation

The State of Vulkan on Apple Devices White paper

richard@lunarg.com


LUNARG®