The Vulkan Profiles Toolset solution

Christophe Riccio
LunarG, Inc

Žiga Markuš
LunarG, Inc
Agenda

- A perspective on why Vulkan Profiles are relevant for Vulkan developers
- Presentation of the Vulkan Profiles Toolset components
- Tutorial on how to use the Vulkan Profiles Toolset components

Based on The Vulkan Profiles Toolset solution whitepaper
Expected audience for this presentation

- Developers who …
  - wrote some code with the Vulkan API.
  - know Vulkan Configurator workflow to setup Vulkan layers.
  - are interested in making sure the Vulkan application will run on end-user platforms.
Designing the Vulkan Profiles Toolset
Creating portable Vulkan Applications in terms of Vulkan capabilities

- **Vulkan Profiles:** *Explicit* Vulkan capability requirements and/or supports.
  - Nothing groundbreaking, just a data convention and a toolset.
  - Not targeting homogeneity of the ecosystem, specifying a domain of relevance.

- *Easier Vulkan development for a selected range of actual ecosystem devices.*
  - Making Vulkan Profiles usable from day one.
Vulkan Profiles usages:

- **Roadmap profiles**: to express guidance on the future direction of Vulkan devices.
- **Platform profiles**: to express the Vulkan support available on different platforms.
- **Device profiles**: to express the Vulkan support of a single Vulkan driver for a Vulkan device.
- **Architecture profiles**: to express the Vulkan support of a class of GPUs.
- **Engine profiles**: to express some rendering code paths requirements of an engine.
- Etc.
Examples of a Platform Profile
Examples of Engine Profiles
The Vulkan Profiles Toolset Components
The Vulkan Profiles Toolset Components

- The Vulkan Profiles schema
- The Vulkan Profiles comparison table
- The Vulkan Profiles layer
- The Vulkan Profiles library

Requires a Vulkan 1.0 driver that supports the VK_KHR_get_physical_device_properties2 extension.

Delivered at BETA development stage.
The Vulkan Profiles Toolset Components

- **The Vulkan Profiles schema**
  - A JSON data format to communicate about Vulkan capabilities:
    - extensions, features, properties, formats, and queue properties.
  - Each revision of Vulkan API is represented by a schema that supersedes older versions of Vulkan API.
- **The Vulkan Profiles comparison table**
- **The Vulkan Profiles layer**
- **The Vulkan Profiles library**
The Vulkan Profiles Toolset Components

- The Vulkan Profiles schema
- The Vulkan Profiles comparison table
  - A markdown representation as a table of Vulkan Profiles to enable comparison.
- The Vulkan Profiles layer
- The Vulkan Profiles library
The Vulkan Profiles Toolset Components

- The Vulkan Profiles schema
- The Vulkan Profiles comparison table
- The Vulkan Profiles layer
  - Downgrade the Vulkan developer’s system capabilities
  - Additional checking of the validity of a profile definition
- The Vulkan Profiles library
The Vulkan Profiles Toolset Components

- **The Vulkan Profiles schema**
- **The Vulkan Profiles comparison table**
- **The Vulkan Profiles layer**
- **The Vulkan Profiles library**
  - A header-only C++ library to use Vulkan Profiles in Vulkan applications.
  - Checking Profiles support on a device and creating a `VkDevice` instance with the profile features and extensions enabled.
Deployment of the Toolset components
Using the Vulkan Profiles schema
Content of Vulkan Profiles JSON files

- A collection of capability sets
  - Capabilities: extensions, features, properties, format properties, queue families properties

- A list of Vulkan Profiles referencing capabilities sets
  - Enable multiple Vulkan Profiles variants per file

- A specification of capabilities and the API we should use, eg:
  - VkPhysicalDeviceDescriptorIndexingFeaturesEXT
  - VkPhysicalDeviceDescriptorIndexingFeatures
  - VkPhysicalDeviceVulkan12Features

- There is a schema per Vulkan Header revision
  - To ensure that we don’t specify in the Profiles file unsupported capabilities for the required Vulkan API revision.
Vulkan Profiles file structure

```json
{
    "$schema": "https://schema.khronos.org/vulkan/profiles-0.8.0-204.json#",
    "capabilities": {
        "baseline": {
            "extensions": {},
            "features": {},
            "properties": {},
            "formats": {},
            "queueFamiliesProperties": []
        },
        "unused": {}
    },
    "profiles": {
        "VP_LUNARG_test_structure_complex": {
            "version": 1, "api-version": "1.2.198",
            "label": "LunarG Profiles Structure Complex unit test",
            "description": "For schema unit test on C.I.",
            "contributors": {},
            "history": [],
            "capabilities": [
                "Baseline"
            ]
        }
    }
}
```
Finding the Vulkan Profiles JSON files

- In the Vulkan SDK package:
  - `${VULKAN_SDK}\Config\VK_LAYER_KHRONOS_profiles`

- In Vulkan Profiles repository:
  - https://github.com/KhronosGroup/Khronos-Schemas/tree/main/vulkan

- On GPUInfo.org website:
  - Eg: https://vulkan.gpuinfo.org/displayreport.php?id=14151
Vulkan Schema Versioning

- profiles-0.8.0-204.json
  - 0.8 => Beta
  - 204 => Vulkan Header version
- profiles-0.8-latest.json
Finding the Vulkan Profiles Schemas

- In the Vulkan SDK package:
  - \${VULKAN_SDK}\share\vulkan\registry

- On Khronos Schemas website (Header 96 to latest):
  - https://schema.khronos.org/vulkan/

- On Khronos Schemas repository (Header 96 to latest):
  - https://github.com/KhronosGroup/Khronos-Schemas/tree/main/vulkan

- On Khronos Profiles repository (*latest.json only):
  - https://github.com/KhronosGroup/Vulkan-Profiles/tree/master/schema
Validating the Roadmap 2022 JSON file. Demo…
Limitations of Vulkan Profiles schema validation

- The schema doesn’t handle duplicated capabilities within a block
  - Eg: `runtimeDescriptorArray` exists in:
    - `VkPhysicalDeviceDescriptorIndexingFeaturesEXT`
    - `VkPhysicalDeviceDescriptorIndexingFeatures`
    - `VkPhysicalDeviceVulkan12Features`
  - The Profiles layer will report messages if such a case occurs.

- The schema doesn’t handle old Vulkan version.
  - Eg: 1.1.204.
  - The Profiles layer will report messages if such a case occurs.

- The schema doesn’t support some capabilities:
  - memory types, surfaces, others?
Using the Vulkan Profiles layer
The Vulkan Profiles layer deployment
Simulate vs Emulate

- Downgrade the developer’s system Vulkan capabilities
- Only emulates `VK_KHR_portability_subset`
- No emulation of mobile specific feature on desktop hardware
Typical testing strategy, a device and driver at a time
Testing Vulkan capabilities support against a Vulkan profile
The Vulkan Profiles layer use cases examples:

- Using C.I. to ensure that the Vulkan application never adds unintentional Vulkan capabilities requirements.
- Verifying the Vulkan application could on a less capable Vulkan device than the Vulkan developer device.
- Verifying that the Vulkan application falls back correctly when a driver doesn’t support a capability without updating the drivers or recompiling the Vulkan application.
- Verifying whether a Vulkan application behavior on a machine is due to the capabilities of that machine.
- Verifying the Vulkan Profile is well formed, with no unexpected duplicated references of Vulkan capabilities.
- Etc.
Using the Profiles layer with Vulkan Configurator. Demo…
Vulkan Profiles layer limitation

- It can’t override Vulkan instance extensions
Using the Vulkan Profiles library
The Vulkan Profiles library deployment
The Vulkan Profiles library API:

- Check Vulkan Profile support by the platform
- Create VkDevice with the profile features and extension enabled
  - Simplifies Vulkan initialization code
- Reflection on the Vulkan Profiles

- The Vulkan Profiles API is not part of the Vulkan API
- Part of a dedicated library
  - Only requires Vulkan 1.0 + VK_KHR_get_physical_device_properties2
  - Delivered with the Vulkan applications
Integration in a Vulkan application

- With a Vulkan SDK dependence
  - `#include <vulkan/vulkan_profiles.hpp>`

- From the [Vulkan-Profiles](https://github.com/Vulkan-Profiles) repository
  - Copy paste in the code base:
    - Either `vulkan_profiles.hpp`
    - Or `vulkan_profiles.h` and `vulkan_profiles.cpp`

```
#define VK_ENABLE_BETA_EXTENSIONS 1
#include <vulkan/vulkan_profiles.hpp>
```
Vulkan Profiles Library API: defines

Example:

```c
#if defined(VK_VERSION_1_3) && defined(VK_KHR_global_priority)
    #define VP_KHR_roadmap_2022 1
    #define VP_KHR_ROADMAP_2022_NAME "VP_KHR_roadmap_2022"
    #define VP_KHR_ROADMAP_2022_SPEC_VERSION 1
    #define VP_KHR_ROADMAP_2022_MIN_API_VERSION VK_MAKE_VERSION(1, 3, 204)
#endif
```

```c
const VpProfileProperties profile = {VP_KHR_ROADMAP_2022_NAME, VP_KHR_ROADMAP_2022_SPEC_VERSION};
```

These defines exist for each Vulkan Profiles implemented by the Profiles Library.
Vulkan Profiles Library API: checking profile support

- VkResult vpGetInstanceProfileSupport(
  const char *pLayerName,
  const VpProfileProperties *pProfile,
  VkBool32 *pSupported);
  ○ Check whether a profile is supported at the instance level

- VkResult vpGetPhysicalDeviceProfileSupport(
  VkInstance instance, VkPhysicalDevice physicalDevice,
  const VpProfileProperties *pProfile,
  VkBool32 *pSupported);
  ○ Check whether a profile is supported by the physical device
Vulkan Profiles Library API: creating a VkDevice

- VkResult vpCreateInstance(
  const VpInstanceCreateInfo *pCreateInfo,
  const VkAllocationCallbacks *pAllocator, VkInstance *pInstance);
  ○ Create a VkInstance with the profile instance extensions enabled

- VkResult vpCreateDevice(
  VkPhysicalDevice physicalDevice,
  const VpDeviceCreateInfo *pCreateInfo,
  const VkAllocationCallbacks *pAllocator, VkDevice *pDevice);
  ○ Create a VkDevice with the profile features and device extensions enabled
Vulkan Profiles Library API: profile reflection

- VkResult vpGetProfileInstanceExtensionProperties(const VpProfileProperties *pProfile,
  uint32_t *pPropertyCount, VkExtensionProperties *pProperties);
  Query the list of instance extensions of a profile

- VkResult vpGetProfileDeviceExtensionProperties(const VpProfileProperties *pProfile,
  uint32_t *pPropertyCount, VkExtensionProperties *pProperties);
  Query the list of device extensions of a profile

- VkResult vpGetProfileFeatureStructureTypes(const VpProfileProperties *pProfile,
  uint32_t *pStructureTypeCount, VkStructureType *pStructureTypes);
  Query the list of feature structure types specified by the profile

- void vpGetProfileFeatures(const VpProfileProperties *pProfile, void *pNext);
  Fill the feature structures with the requirements of a profile

- VkResult vpGetProfilePropertyStructureTypes(const VpProfileProperties *pProfile,
  uint32_t *pStructureTypeCount, VkStructureType *pStructureTypes);
  Query the list of property structure types specified by the profile

- void vpGetProfileProperties(const VpProfileProperties *pProfile, void *pNext);
  Fill the property structures with the requirements of a profile
Vulkan Profiles Library API: profile reflection

- VkResult vpGetProfileQueueFamilyProperties(const VpProfileProperties *pProfile,
  uint32_t *pPropertyCount, VkQueueFamilyProperties2KHR *pProperties);
  - Query the requirements of queue families by a profile

- VkResult vpGetProfileQueueFamilyStructureTypes(const VpProfileProperties *pProfile,
  uint32_t *pStructureTypeCount, VkStructureType *pStructureTypes);
  - Query the list of query family structure types specified by the profile

- VkResult vpGetProfileFormats(const VpProfileProperties *pProfile,
  uint32_t *pFormatCount, VkFormat *pFormats);
  - Query the list of formats with specified requirements by a profile

- void vpGetProfileFormatProperties(const VpProfileProperties *pProfile,
  VkFormat format, void *pNext);
  - Query the requirements of a format for a profile

- VkResult vpGetProfileFormatStructureTypes(const VpProfileProperties *pProfile,
  uint32_t *pStructureTypeCount, VkStructureType *pStructureTypes);
  - Query the list of format structure types specified by the profile
Vulkan Profiles Library API: Listing profiles

- VkResult vpGetProfiles(uint32_t *pPropertyCount, VpProfileProperties *pProperties);
  - Query the list of available profiles in the library

- VkResult vpGetProfileFallbacks(const VpProfileProperties *pProfile,
  uint32_t *pPropertyCount, VpProfileProperties *pProperties);
  - List the recommended fallback profiles of a profile
Vulkan Profiles library sample

A Vulkan sample is available for demonstrating Vulkan Profiles library usage.
Using the Profiles library in test_profile_example.cpp. Demo...
The Vulkan Profiles Toolset code generation
Exercise: a Vulkan 1.2 roadmap 2022 profile

"VP_KHR_roadmap_2022_1_2": {
  "version": 1,
  "api-version": "1.2.197",
  "label": "Khronos Roadmap 2022 profile on Vulkan 1.2",
  "description": "This roadmap profile is intended to implement the Vulkan Roadmap 2022 profile on Vulkan 1.2."
  "capabilities": [
    "vulkan10requirements",
    "vulkan10requirements_roadmap2022",
    "vulkan11requirements",
    "vulkan11requirements_roadmap2022",
    "vulkan12requirements",
    "vulkan12requirements_roadmap2022",
    "vulkan13requirements_1_2",
    "vulkan13requirements_roadmap2022_1_2"
  ]
}
Exercise: a Vulkan 1.2 roadmap 2022 profile

```json
"vulkan13requirements_roadmap2022": {
   "extensions": {
      "VK_KHR_global_priority": 1
   },
   "features": {
      "VkPhysicalDeviceVulkan13Features": {
         "descriptorBindingInlineUniformBlockUpdateAfterBind": true
      }
   }
},
"vulkan13requirements_roadmap2022_1_2": {
   "extensions": {
      "VK_EXT_global_priority": 1,
      "VK_EXT_inline_uniform_block": 1
   },
   "features": {
      "VkPhysicalDeviceInlineUniformBlockFeaturesEXT": {
         "descriptorBindingInlineUniformBlockUpdateAfterBind": true
      }
   }
}
```
Rebuilding the repository with additional Vulkan Profiles. Demo…
Future Improvement:
Improving Vulkan Profiles creation
Improving Vulkan Profiles creation

- **Vulkaninfo** export of Device profiles JSON files (PR)
- Tool to combine profiles
  - Either accumulating the requirements (For Engine profiles)
  - Or subtracting available support (For Platform profiles)
- Tool to validate a profile against another
- Better handling of platform specific capabilities
  - Surface extensions are necessary but platform specific
- Add more Vulkan capabilities to the schema (memory types, surfaces?)
- Improvements on vk.xml:
  - Default values for capabilities, improved limit types, etc
- Vulkan Configurator integration?
Thanks

Q&A?

Christophe Riccio
LunarG, Inc

Žiga Markuš
LunarG, Inc