ДОДАТОК А

Графічний матеріал атестаційної роботи

Модель кросплатформного програмного модуля **VulkanKit**

Мельніченко Федір, СПзм-18-2

Мета роботи

Створення моделі для взаємодії з графічним драйвером у вигляді програмного інтерфейсу (API)

Створення **моделі програмного інтерфейсу** та **реалізація у вигляді програмного модуля** на базі існуючих технологій, відповідаючи вимогам ринку і є **головною метою досліджень** та **розробки (R&D)** данного проекту.

Аналіз аналогів

Провести аналіз існуючих аналогів чи близько подібних реалізацій. Взяти до уваги переваги та недоліки існуючих імплементацій.

Проектування API

Розробити дизайн нового програмного інтерфейсу, враховуючи переваги існуючих імплементацій + **побажання розробників**.

Імплементація API

Імплементування та тестування API у вигляді кросплатформного програмного модулю на C++.

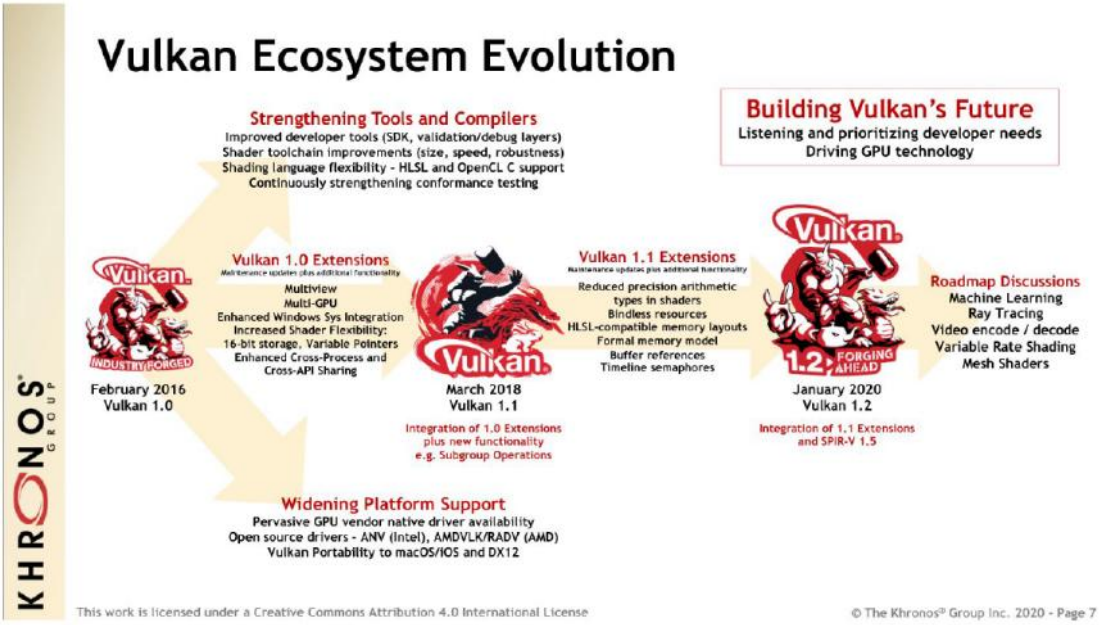
Vulkan API

Кросплатформний інтерфейс програмування нового покоління (API) призначений для графічних і обчислювальних пристроїв.

KHRONOS GROUP

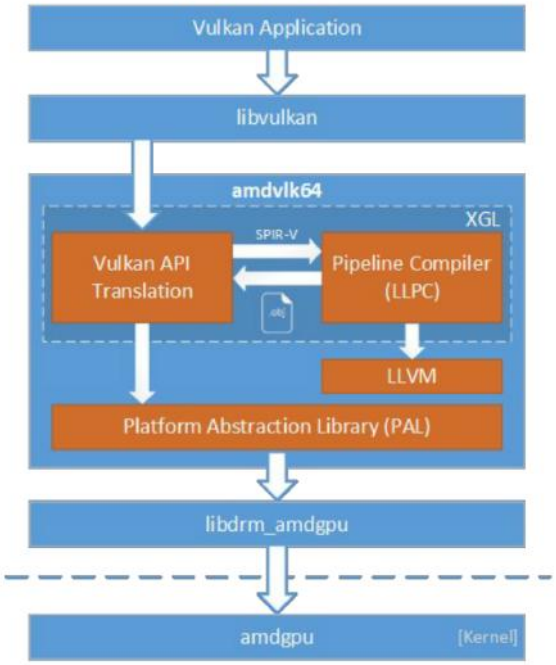
Vulkan

THE FUTURE OF GRAPHICS



Vulkan ecosystem

Vulkan & Driver



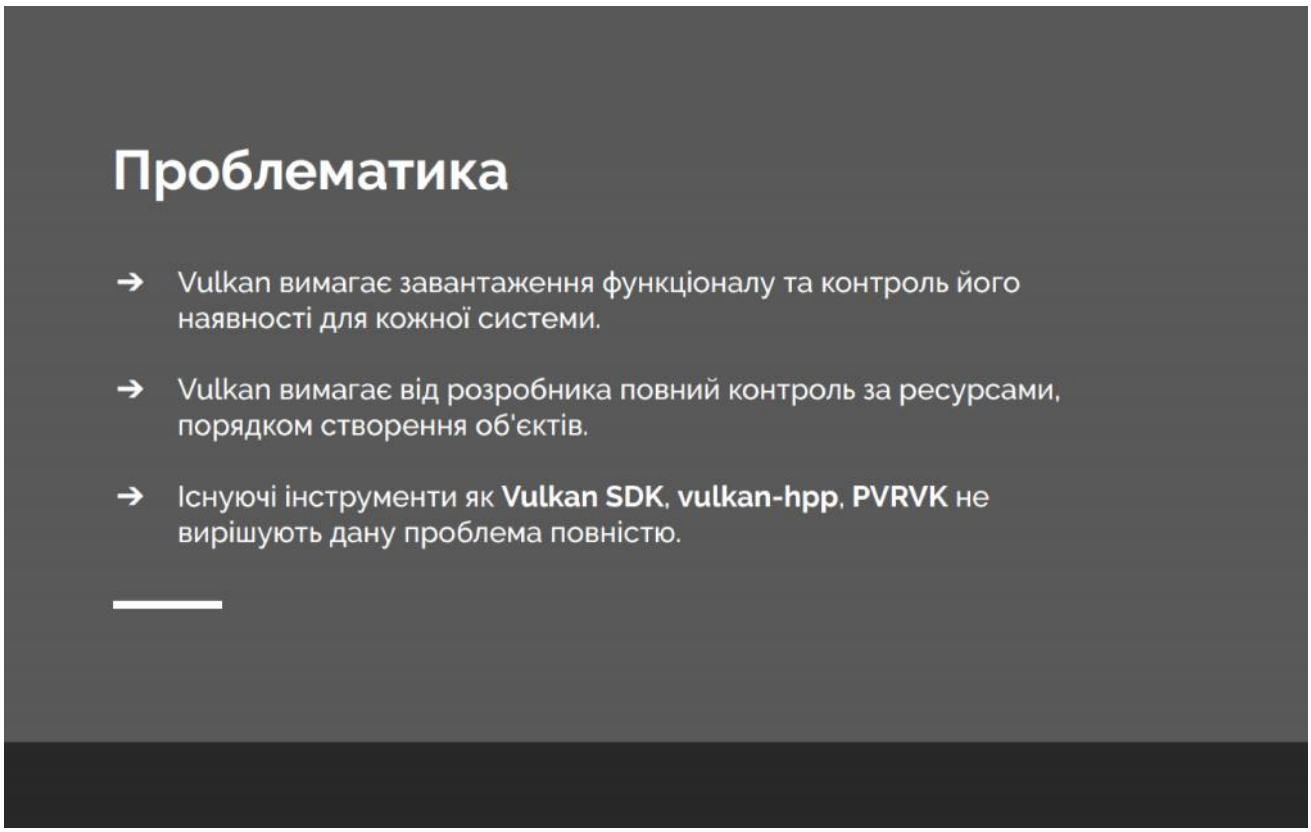
Vulkan

- Vulkan - програмний інтерфейс драйвера нового покоління
- Використовується в багатьох галузях розробки програмного забезпечення
- Явний (explicit) програмний інтерфейс вимагає особливої уваги під час розробки

Код програми, котра виводить інформацію кожного **графічного адаптера** в системі (**ім'я, версія драйвера, індекс вендора**)

~**80** рядків коду :(

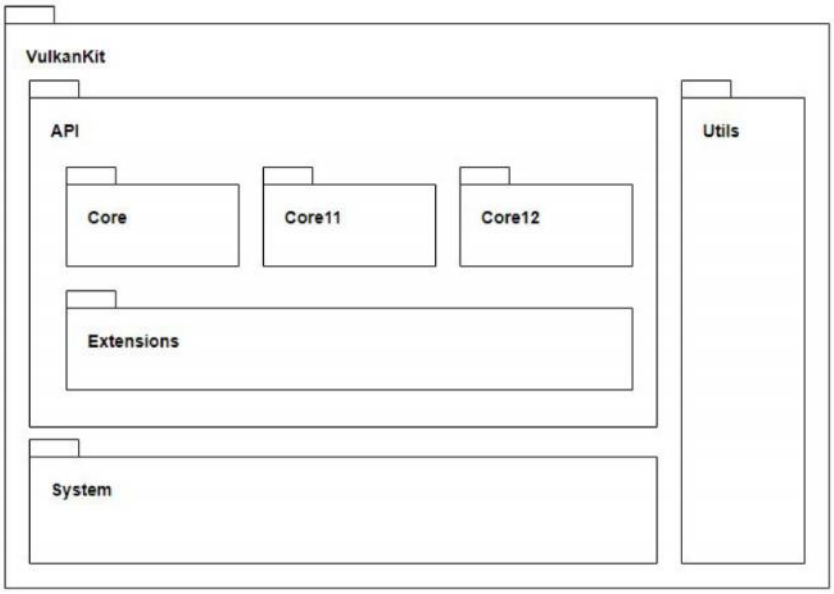
```
#include <windows.h>

#define VK_USE_PLATFORM_WIN32_KHR
#include <vulkan/vulkan.h>

int main(int argc, char** argv)
{
    auto vulkanModule = ::LoadLibraryA("vulkan-1.dll");
    assert(vulkanModule);

    auto pfnGetInstanceProcAddr = (::PFN_vkGetInstanceProcAddr)::GetProcAddress(vulkanModule, "vkGetInstanceProcAddr");
    auto pfnCreateInstance = (::PFN_vkCreateInstance)pfnGetInstanceProcAddr(nullptr, "vkCreateInstance");
    auto pfnEnumerateInstanceVersion = (::PFN_vkEnumerateInstanceVersion)pfnGetInstanceProcAddr(nullptr, "vkEnumerateInstanceVersion");

    const char* extensions[] = { "VK_KHR_get_physical_device_properties2", "VK_KHR_driver_properties" };

    uint32_t instanceApiVersion = VK_MAKE_VERSION(1, 0, 0);
    if (pfnEnumerateInstanceVersion)
    {
        pfnEnumerateInstanceVersion(&instanceApiVersion);
    }

    ::VkApplicationInfo applicationInfo{};
    applicationInfo.apiVersion = instanceApiVersion;

    ::VkInstanceCreateInfo instanceCreateInfo{};
    instanceCreateInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
    instanceCreateInfo.pApplicationInfo = &applicationInfo;
    instanceCreateInfo.ppEnabledExtensionNames = extensions;
    instanceCreateInfo.enabledExtensionCount = ::std::size(extensions);

    ::VkInstance instance{ nullptr };
    pfnCreateInstance(&instanceCreateInfo, nullptr, &instance);
    assert(instance);

    auto pfnEnumeratePhysicalDevices = (::PFN_vkEnumeratePhysicalDevices)pfnGetInstanceProcAddr(instance, "vkEnumeratePhysicalDevices");
    auto pfnGetPhysicalDeviceMemoryProperties2 = (::PFN_vkGetPhysicalDeviceMemoryProperties2)
            pfnGetInstanceProcAddr(instance, "vkGetPhysicalDeviceMemoryProperties2");
    if (!pfnGetPhysicalDeviceMemoryProperties2)
    {
        pfnGetPhysicalDeviceMemoryProperties2 = (::PFN_vkGetPhysicalDeviceMemoryProperties2)
                pfnGetInstanceProcAddr(instance, "vkGetPhysicalDeviceMemoryProperties2KHR");
    }
    assert(pfnGetPhysicalDeviceMemoryProperties2);

    uint32_t physicalDeviceCount{};
    pfnEnumeratePhysicalDevices(instance, &physicalDeviceCount, nullptr);
    ::std::vector<::VkPhysicalDevice> physicalDevices{ physicalDeviceCount };
    pfnEnumeratePhysicalDevices(instance, &physicalDeviceCount, physicalDevices.data());

    for (auto index{ 0u }; index != physicalDeviceCount; ++index)
    {
        ::VkPhysicalDeviceDriverProperties driverProperties{};
        driverProperties.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DRIVER_PROPERTIES;

        ::VkPhysicalDeviceProperties2 properties{};
        properties.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROPERTIES_2;
        properties.pNext = &driverProperties;

        pfnGetPhysicalDeviceProperties2(physicalDevices[index], &properties);

        ::std::clog << "GPU[" << index << "]: "
            << properties.properties.deviceName << ", "
            << driverProperties.driverName << " "
            << driverProperties.driverInfo << ", "
            << properties.properties.vendorID << "\n";
    }

    auto pfnDestroyInstance = (::PFN_vkDestroyInstance)pfnGetInstanceProcAddr(nullptr, "vkDestroyInstance");
    pfnDestroyInstance(instance, nullptr);

    ::FreeLibrary(vulkanModule);
}
```


Проблематика
Vulkan вимагає завантаження функціоналу та контроль його наявності для кожної системи.
Vulkan вимагає від розробника повний контроль за ресурсами, порядком створення об'єктів.
Існуючі інструменти як Vulkan SDK, vulkan-hpp, PVRVK не вирішують дану проблема повністю.

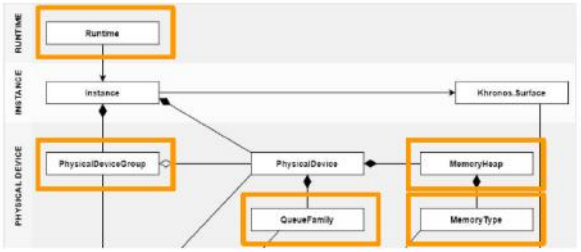

VulkanKit
VulkanKit API - об'єктно-орієнтований програмний інтерфейс для взаємодії з графічним драйвером через низькорівневий функціонал Vulkan
Vulkan API
+
OOD
=
VulkanKit API

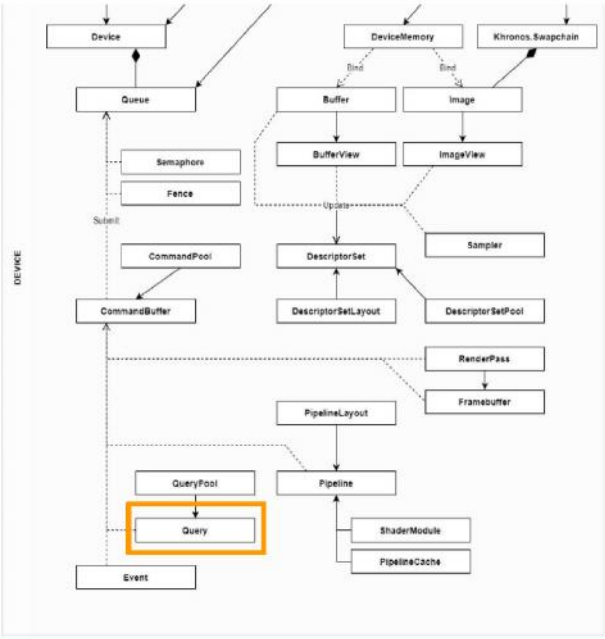
Вимоги до моделі VulkanKit

- Об'єктно-орієнтована модель
- Дотримання об'єктної моделі та моделі виконання Vulkan
- Багаторівневий дизайн компонентів у вигляді орієнтованого ациклічного графу
- Модульність

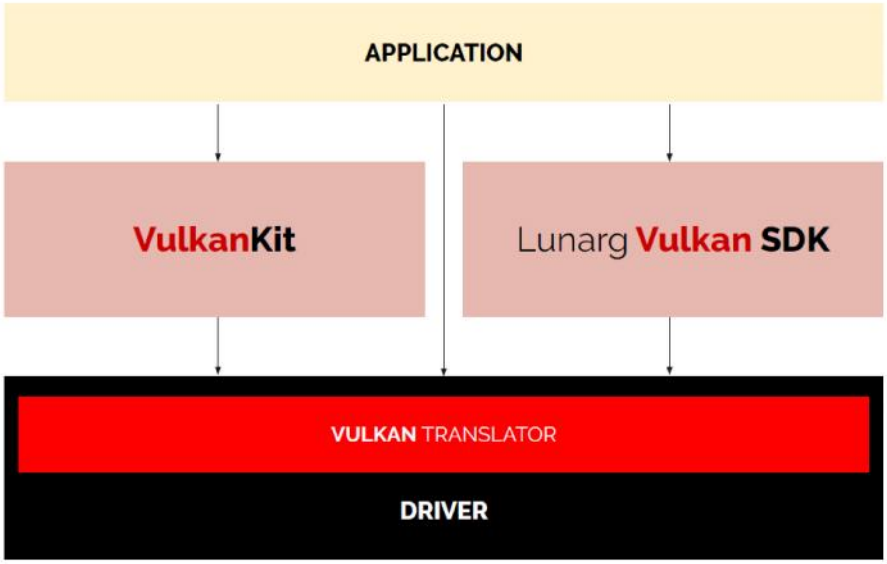


VulkanKit API - Package Model

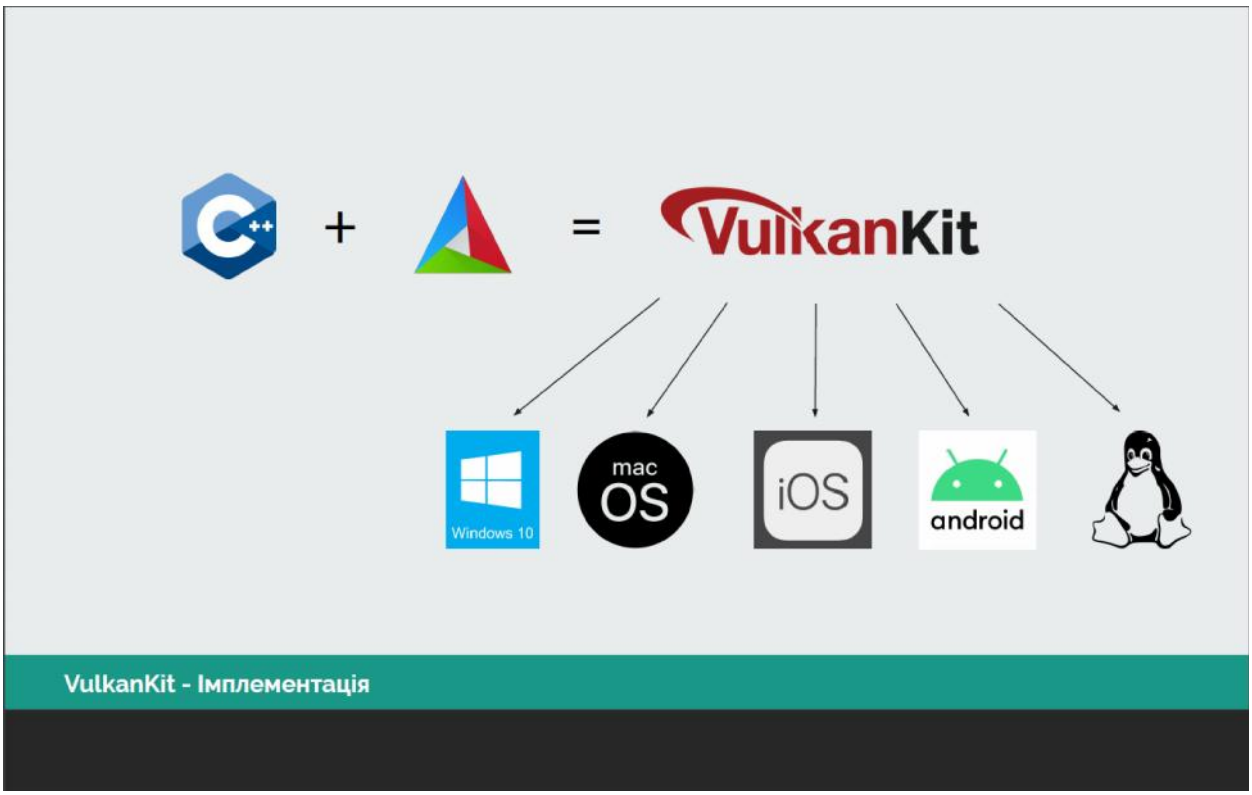




VulkanKit API - Component Model



VulkanKit



VulkanKit - Імплементація

```cpp
#include <Windows.h>

#define VK_USE_PLATFORM_WIN32_KHR
#include <vulkan/vulkan.h>

int main(int argc, char** argv)
{
    auto vulkanModule = ::LoadLibrary("vulkan-1.dll");
    assert(vulkanModule);
    auto pfnGetInstanceProcAddr = (::PFN_vkGetInstanceProcAddr)::GetProcAddress(vulkanModule, "vkGetInstanceProcAddr");
    auto pfnCreateInstance = (::PFN_vkCreateInstance)pfnGetInstanceProcAddr(nullptr, "vkCreateInstance");
    auto pfnEnumerateInstanceVersion = (::PFN_vkEnumerateInstanceVersion)pfnGetInstanceProcAddr(nullptr, "vkEnumerateInstanceVersion");

    const char* extensions[] = { "VK_KHR_get_physical_device_properties2", "VK_KHR_driver_properties" };

    uint32_t instanceApiVersion = VK_MAKE_VERSION(1, 0, 0);
    if (pfnEnumerateInstanceVersion)
    {
        pfnEnumerateInstanceVersion(&instanceApiVersion);
    }

    ::VkApplicationInfo applicationInfo{};
    applicationInfo.apiVersion = instanceApiVersion;
    ::VkInstanceCreateInfo instanceCreateInfo{};
    instanceCreateInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
    instanceCreateInfo.pApplicationInfo = &applicationInfo;
    instanceCreateInfo.ppEnabledExtensionNames = extensions;
    instanceCreateInfo.enabledExtensionCount = ::std::size(extensions);

    ::VkInstance instance{ nullptr };
    pfnCreateInstance(&instanceCreateInfo, nullptr, &instance);
    assert(instance);

    auto pfnEnumeratePhysicalDevices = (::PFN_vkEnumeratePhysicalDevices)pfnGetInstanceProcAddr(instance, "vkEnumeratePhysicalDevices");
    auto pfnGetPhysicalDeviceMemoryProperties2 = (::PFN_vkGetPhysicalDeviceMemoryProperties2)pfnGetInstanceProcAddr(instance, "vkGetPhysicalDeviceMemoryProperties2");
    if (!pfnGetPhysicalDeviceMemoryProperties2)
    {
        pfnGetPhysicalDeviceMemoryProperties2 = (::PFN_vkGetPhysicalDeviceMemoryProperties2)
            pfnGetInstanceProcAddr(instance, "vkGetPhysicalDeviceMemoryProperties2KHR");
    }
    assert(pfnGetPhysicalDeviceMemoryProperties2);

    uint32_t physicalDeviceCount{};
    pfnEnumeratePhysicalDevices(instance, &physicalDeviceCount, nullptr);
    ::std::vector<::VkPhysicalDevice> physicalDevices{ physicalDeviceCount };
    pfnEnumeratePhysicalDevices(instance, &physicalDeviceCount, physicalDevices.data());

    for (auto index{ 0u }; index != physicalDeviceCount; ++index)
    {
        ::VkPhysicalDeviceDriverProperties driverProperties{};
        driverProperties.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DRIVER_PROPERTIES;
        ::VkPhysicalDeviceProperties2 properties{};
        properties.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROPERTIES_2;
        properties.pNext = &driverProperties;

        pfnGetPhysicalDeviceProperties2(physicalDevices[index], &properties);

        ::std::clog << "GPU[" << index << "]: "
            << properties.properties.deviceName << ", "
            << driverProperties.driverName << " "
            << driverProperties.driverInfo << ", "
            << properties.properties.vendorID << "\n";
    }

    auto pfnDestroyInstance = (::PFN_vkDestroyInstance)pfnGetInstanceProcAddr(nullptr, "vkDestroyInstance");
    pfnDestroyInstance(instance, nullptr);

    ::FreeLibrary(vulkanModule);
}
```

```cpp
#include <VulkanKit/VulkanKit.hpp>

int main(int argc, char** argv)
{
  using namespace vulkankit::API;

  auto runtime = core::CreateRuntime({});
  VulkanKitAssert(runtime);

  auto instance = runtime->CreateInstance({});
  VulkanKitAssert(instance);

  for (const auto physicalDevice : instance->GetPhysicalDevices())
  {
    Core::PhysicalDeviceProperties properties{},
    physicalDevice->GetProperties(properties);

    const auto extendedPhysicalDevice = physicalDevice->As<Extensions::DriverProperties::PhysicalDevice>();

    Extensions::DriverProperties::DriverProperties driverProperties{};
    extendedPhysicalDevice->GetDriverProperties(driverProperties);

    ::std::clog << "GPU: " << properties.name << ", "
        << driverProperties.driverName << " "
        << driverProperties.driverInfo<< ", "
        << properties.vendor << "\n";
  }
}
```

Vulkan - ~**80** рядків коду :(

VulkanKit - ~**30** рядків коду :)

VulkanKit

Висновок

- Розроблено модель взаємодії з графічним адаптером
- На базі даної моделі спроектовано програмний інтерфейс VulkanKit API
- Проведено дослідження властивостей ОО мов програмування
- Проведено дослідження в галузі проектування програмного забезпечення
- Імплементовано першу тестову реалізацію VulkanKit (C++/CMake)

Перспективи

- **Вихід в open-source простір (CPP + CMake).**
- Підтримка нових платформ (Google Stadia, Nintendo, тощо)
- Реалізація для інших мов програмування (Rust, C#, Swift)
- Розробка ABI compatible версії *