Chapter 1 소개

튜토리얼에 대하여…

이 튜토리얼은 Vulkan 그래픽스와 Compute API에 대해 가르칠 것입니다. Vulkan은 Khronos 그룹의 새로운 API로 현대(Modern) 그래픽 카드에게 더 향상된 추상화(Abstraction)을 제공할 것입니다. 이 새로운 인터페이스는 당신의 응용 프로그램이 더 좋은 성능을 이끌어내고, OpenGL과 Direct3D와 같은 API와 비교해서 덜 예외적인(Less surprising) 행동을 할 수 있도록 작성하기 쉽도록 제공되고 있습니다. 이러한 Vulkan의 아이디어는 Direct3D 12와 Metal과 비슷할 수 있습니다. 하지만 Vulkan은 완전한 크로스 플랫폼으로써 Windows, Linux 그리고 Android에서도 동시에 개발할 수 있다는 점에서 장점(Advantage)을 가지게 됩니다.

하지만 이러한 장점(benefits)를 취하기 위해서는 굉장히 장황한 API작업을 해야 합니다. 당신이 급조한 응용 프로그램에서도 프레임 버퍼 생성 초기화, 버퍼나 텍스처 이미지와 같은 메모리 관리자와 같은 그래픽 API와 관련해서 상세하게 작성(detail)해야 합니다. 그래픽 드라이버에 의존하는(less hand-holding)게 많이 줄어들기 때문에 올바른 행동을 할 수 있도록 더 많은 작업을 해야 합니다.

여기서 전달하는 메시지는 Vulkan은 모두를 위한 API가 아니라는 것입니다. 이는 고성능의 컴퓨터 그래픽에 열광하는(enthusiastic) 프로그래머와 기꺼이 그렇게 하려는 사람(are willing to put some work in)을 타겟으로 잡고 있습니다. 만약에 당신이 컴퓨터 그래픽스보다 게임 개발에 더 흥미가 있다면 Vulkan에 대한 흥미가 곧 떨어져 OpenGL이나 Direct3D을 다시 선호하게 될 수 있습니다. 다른 방법이라면 Unreal 엔진이나 유니티는 Vulkan을 사용할 수 있을 뿐만 아니라 더 높은 API를 제공할 것입니다.

이걸로 이제 이 튜토리얼을 따라오기 위한 준비물을 공개하겠습니다.

* Vulkan과 호환되는 그래픽 카드와 드라이버 (NVIDIA, AMD, Intel)
* C++에 대한 경험 (RAII와 초기화 리스트에 친숙)
* C++ 11이 호환되는 컴파일러 ( Visual Studio 2013 이상, GCC 4.8 이상)
* 3D 컴퓨터 그래픽스에 대한 어느정도의 경험

이 튜토리얼에서는 OpenGL이나 Directe3D의 개념(Concepts)에 대한 얘기는 하지 않을 것입니다. 그러나, 3D 컴퓨터 그래픽스에 대한 기초 지식은 갖추고 있어야 합니다. 원근 투영과 같은 수학적 배경을 설명하지 않기 때문에 컴퓨터 그래픽스 개념에 대해서는 온라인 자습서를 참조하는 게 좋습니다.

원한다면 C++ 대신 C를 사용할 수 있지만, 예제와 다른 선형 대수 라이브러리를 사용해야 하고, 본인만의 코드를 작성해야 합니다. 우리는 로직과 자원의 생존주기(Lifetime)을 설명하기 위해서 C++의 클래스와 RAII와 같은 기능을 이용할 것입니다.

E-book

E-book으로 이 튜토리얼을 읽고 싶으면 EPUB파일이나 PDF 파일을 다운로드 받을 수 있습니다.

* EPUB
* PDF

튜토리얼 내용(Structure)

우리는 첫 번째 삼각형을 화면에 띄우는 방법과, Vulkan의 작동 방식에 대한 개요로 시작하겠습니다. 모든 작은 단계의 목적은 당신이 전체적인 그림 안에서 기본적인 역할을 이해할 수 있도록 할 것입니다. 다음으로, Vulkan SDK와 선형 대수 계산을 위한 GLM 라이브러리, 창 생성을 위한 GLFW로 개발 환경을 설정할 것입니다. 튜토리얼은 Windows에서 Visual Studio, Ubuntu Linux에서 GCC를 이용한 설정 방법을 알려 줄 것입니다.

그리고 삼각형을 그리는데 필요한 Vulkan 프로그램의 요소들을 모두 구현할 것입니다. 각 챕터는 간략하게 다음과 같은 구조를 따르게 됩니다.

* 새로운 개념에 대한 소개와 목적 소개
* 관련 API를 모두 호출하고 프로그램에 통합
* 추상화 된 헬퍼 함수 작성

각 챕터는 이전 챕터의 후속 내용으로 작성이 되어있지만, Vulkan 기능을 소개하고 알아볼 수 있는 독립 챕터로써 읽을 수도 있습니다. 이는 사이트에서 유용한 참고 자료로 이용할 수도 있습니다. 모든 Vulkan 함수와 타입들은 사양서(Specification)에 링크되어 있으니, 더 알아보시고 싶으시면 클릭하시면 됩니다. Vulkan은 매우 새로운 API입니다. 그래서 사양서에 문제점(Shortcomings)가 있을 수 있습니다. 문제점을 발견하면 Khronos 저장소에 피드백을 남겨주시길 바랍니다.

이전에 언급했듯이, Vulkan API는 그래픽스 하드웨어를 최대한 통제하기 위해 많은 인자(Paramter)가 포함된 장황한 API를 가지고 있습니다. 이는 간단한 텍스처 생성과 같은 작업에도 많은 단계를 거치고, 매번마다 반복해야 하는 작업을 야기할 수 있습니다. 그러므로 우리는 우리만의 튜토리얼에서 사용할 헬퍼 함수들을 작성할 것입니다.

모든 챕터는 코드가 작성되어 있는 링크를 포함합니다. 만약에 당신이 코드 구조에 대해서 궁금증(Doubts)이 들거나, 디버깅 또는 비교를 하고 싶을 때 알아볼 수 있습니다. 모든 코드 파일은 다양한 인증된 공급자(Vendors)에서 테스트되었습니다. 또한 각 챕터의 마지막에서는 특정 주제에 대해서 질문을 할 수 있는 섹션이 있습니다. 플랫폼과, 드라이버 버전, 소스코드 그리고 예상했던 행동과 실제 행동을 작성하면 도움을 드리겠습니다.

이 튜토리얼은 커뮤니티의 영향을 위해서 작성되었습니다. Vulkan은 여전히 새로운 API이며, 대표적으로 좋은 예제는 아직 없습니다. 만약에 튜토리일과 사이트에 대해서 피드백이 있다면 망설이지 마시고 GitHub 저장소에 issuse나 pull request를 남겨주시길 바랍니다. 저장소에 Watch를 해두신다면 튜토리얼 업데이트에 대한 알림을 받으실 수 있습니다.

당신이 Vulkan의 삼각형을 화면에 띄우는 절차를 마친 후에, 선형 변환을 포함한 텍스처, 3D 모델작업을 시작할 것입니다.

이전에 그래픽스 API를 다뤄보았다면 첫 번째 지오메트리를 띄우는데 많은 작업이 있음을 알 수 있습니다. Vulkan에는 많은 초기화 단계들이 있지만, 각각의 단계들이 이해하기 쉬울것이고, 중복되지 않다고 느낄 것입니다. 또한 보이는게 지루한 삼각형을 띄우고 나면 완벽히 그려진 3D 모델을 띄우는데 큰 노력이 들지 않으며, 각 단계를 넘을 때마다 보람참을 느끼는 게 중요합니다.

만약에 튜토리얼을 따라오던 중 문제를 맞닥뜨리게 되면, 첫 번째로 FAQ에서 당신의 문제와 해결법이 있는지 확인해보십시오. 이후에도 문제가 계속 된다면 가장 가까운 챕터의 질문 섹션에서 편하게 도움을 요청하세요.

고성능 그래픽스 API에 뛰어들 준비가 되셨나요? Let’s go!

Chapter 2 개요

이번 챕터에서는 Vulkan에 대한 소개와 현 시점의 문제점을 소개하고 끝마칠 것입니다. 그리고 나선 첫 번째 삼각형을 그리는데 필요한 구성 요소들을 살펴볼 것입니다. 이러한 방식으로 다음 챕터에 대한 큰 그림을 그릴 수 있도록 합니다. 우리는 Vulkan API의 구조와 일반적인 사용 패턴 등을 알아보고 끝을 맺을 것입니다.

Vulkan의 기원

다른 이전의 그래픽스 API와 비슷하게 Vulkan은 GPU에서 추상화된 크로스 플랫폼 API로 디자인되었습니다. 이 시대에서 디자인된 기능적 그래픽스 하드웨어의 대부분의 API들의 문제점은 구성 가능한(Configurable) 고정 기능(Functionality)에 제한된다는 점입니다. 프로그래머는 표준 포맷내의 버텍스 데이터를 제공해야 하고, GPU 제조사에서 제공하는 라이팅과 셰이딩 옵션에 감사(Mercy)해야할 따름입니다.

그래픽 카드 아키텍처가 성장(Matured)해갈수록 점점 더 많은 프로그래밍 가능한 기능들을 제공하기 시작할 것입니다. 이러한 모든 새로운 기능들은 어떻게 든 현재의 API에 통합해야 합니다. 이러한 결과는 불완전한 추상화를 낳을 수 있으며, 현대의 그래픽스 아키텍처에 대한 프로그래머의 의도에 대해서 그래픽 드라이버 단에서 많은 의혹(Guesswork)을 남기게 될 것입니다. 이러한 이유 때문에 게임의 성능을 향상시키기 위해 많은 드라이버 업데이트가 있습니다, 때때로는 상당한 기간을 두고 업데이트 하기도 합니다. 이런 드라이버들의 복잡성 때문에 응용 프로그램 개발자들은 셰이더를 위한 문법과 같은 공급자 간의 불일치성을 해결해야 합니다. 이러한 새로운 특징 외에도, 지난 10년 동안 강력한 그래픽 하드웨어를 갖춘 모바일 기기들이 쏟아져 나왔다. 이러한 모바일 GPU들은 각기 다른 에너지 그리고 공간 요구 조건에 대한 아키텍처를 기반으로 잡고 있습니다. 한 가지 예제를 뽑자면 타일 렌더링과 같은 기능을 프로그래머에게 더 제공함으로써 더 좋은 성능을 이끌어 낼 수 있도록 하는 것입니다. 제한된 멀티 쓰레딩 지원을 하는 API가 유래하는 또 다른 제약으로는, CPU 단에서 병목 현상이 일어날 수 있다는 점입니다.

Vulkan은 현대 그래픽스 아키텍처들을 위해서 이러한 문제점을 해결할 수 있도록 처음부터 설계되었습니다. 이는 프로그래머가 장황한 API를 사용해 작업에 대해서 깔끔하게 명시하여 드라이버 오버헤드를 줄일 수 있으며, 멀티 쓰레드를 사용해 명령을 동시에(parallel) 생성하고 입력할 수 있도록 합니다. 셰이더 컴파일에서 단일 컴파일러를 이용해 표준화된 바이트 코드 포맷으로 치환함으로써 불일치성을 줄일 수 있습니다. 마지막으로 그래픽스와 컴퓨트 기능을 하나의 API로 통합함으로 현대 그래픽 카드의 범용 계산 능력을 신뢰할 수 있습니다.

삼각형을 그리기 위해 필요한 것

이제 우리는 잘 작동하는 삼각형 그리기 Vulkan 프로그램의 모든 단계의 개요를 살펴볼 것입니다. 여기서 소개된 모든 개념들은 다음 챕터에서 상세하게 설명이 될 것입니다. 이번은 그냥 각각의 요소들에 대해서 큰 그림을 그릴 수 있도록 해드립니다.

Step 1 – 인스턴스와 물리 장치(Physical device) 선정

Vulkan 응용 프로그램은 **VkInstance**라는 Vulkan API를 설정함으로써 시작됩니다. 인스턴스는 당신의 응용 프로그램에 대한 설명과 사용할 확장 API을 기술함으로써 생성됩니다. 인스턴스를 생성한 후에는, Vulkan 지원 하드웨어를 질의(Query)할 수 있고, 명령을 실행하기 위한 하나의 또는 더 많은 **VkPhysicalDevices**를선택할 수 있습니다.원하는 장치들을 선택하기 위해 VRAM 크기와 장치 사양(Capabilities)와 같은 속성들을 질의할 수 있습니다.

Step 2 – 논리 장치(Logical device)와 큐 패밀리

사용할 하드웨어 장치를 선택하고 난 후, VkDevice를 생성해야 합니다. 여기서는VkPhysicalDeviceFeatures에서 다중 뷰포트 렌더링과 64bit 부동 소수점과 같은 사용할 기능들을 구체적으로 서술합니다. 또한 여기서 당신이 사용하고 싶은 큐 패밀리를 지정해야합니다. 그리기 명령이나 메모리 연산과 같은 대부분의 연산들은 VkQueue에 제출함으로써 비동기 시간적으로 실행됩니다. 큐들은 큐 패밀리로부터 할당되며, 각각의 큐 패밀리는 특정 집합의 명령어들을 각 큐들에게 지원할 수 있도록 합니다. 예를 들어, 그래픽스, 컴퓨트 그리고 메모리 전달 연산에 대해서 각각의 큐 패밀리가 존재할 수 있습니다. 큐 패밀리에 대한 유용성은 물리 장치 선정에서 다른(Distinguishing) 인자를 가지게 되는 곳에서도 사용될 수 있습니다. 이는 Vulkan을 지원하는 장치가 그래픽 기능을 지원하지 않을 수도 있지만, 오늘날의 Vulkan을 지원하는 모든 그래픽 카드는 우리가 관심있는(interested) 큐 연산을 지원할 것입니다.

Step 3 – 창 표면(Window surface)와 스왑 체인

당신이 화면에 나타나지 않는(Offscreen) 렌더링에 관심이 있는 게 아니라면, 렌더된 이미지를 표현하는 창을 생성할 필요가 있습니다. Windows는 자체 지원(Native) 플랫폼 API나 GLFW와 SDL과 같은 라이브러리를 이용해 생성할 수 있습니다. 우리는 GLFW를 사용할 것입니다, 자세한 내용은 다음 챕터에서 알아볼 것입니다.

우리는 창에 실제로 렌더하기 위한 창 표면(a window surface)(VkSurfaceKHR)과 스왑 체인(a swap chain)(VkSwapChainKHR)라는 2가지 요소가 더 필요합니다. KHR 접두어에 대해서 얘기하자면 이러한 오브젝트들은 Vulkan 확장의 일부라는 뜻을 가지고 있습니다. Vulkan API는 스스로 완벽하게 특정 플랫폼에서 벗어나 있기 때문에 창 관리자(window manager)와 상호 작용하기 위해 표준화된 WSI(Window System Inteface) 확장을 이용해야 하는 이유입니다. 표면은 렌더링할 창들에 대해서 크로스 플랫폼으로 추상화되었으며, 일반적으로 Windows의 HWND와 같이 Native 창 핸들의 참조를 받아 인스턴스화 됩니다. 운 좋게도, GLFW 라이브러리는 이러한 플랫폼 세부 사항을 다루는 내장 함수가 존재합니다.

스왑 체인은 렌더 타겟들의 집합(Collection)입니다. 이것의 기본적인 목적은 현재 화면에서 렌더링 되고 있는 것과 우리가 렌더링 해야 할 이미지와 다른 것인지 확인하는 것입니다. 이는 완벽한 이미지만 보이게 하기 위한 중요한 작업입니다. 프레임을 그릴 때 마다 스왑 체인에게 렌더링할 이미지를 제공받아야 합니다. 프레임을 그리는게 끝냈다면, 특정한 시점에 화면에 표현하기 위해서 이미지를 스왑 체인에게 다시 반환합니다. 렌더 타겟의 수와 완성된 이미지를 화면에 표현하기 위한 조건은 현재 표현 모드(Present Mode)에 따라 결정됩니다. 일반적인 표현 모드는 이중 버퍼링(vsync) 그리고 삼중 버퍼링입니다. 우리는 이것들을 스왑 체인 생성 챕터에서 살펴볼 것입니다.

Step 4 – 이미지 뷰 그리고 프레임버퍼

스왑 체인으로부터 획득한 이미지를 그리기 위해서는 먼저 VkImageView와 VkFramebuffer로 감싸야 합니다. 이미지 뷰는 이미지에서 사용할 특정 부분을 참조하며, 프레임 버퍼는 이미지 뷰에서 사용할 색상, 깊이, 스텐실 타겟에 대해서 참조합니다. 왜냐하면 스왑 체인 내에는 다른 이미지가 많이 존재할 수 있기 때문에, 각각 이미지에 대해서 우선적으로 이미지 뷰와 프레임 버퍼를 생성하고, 이미지를 그릴 때 맞는 것을 선택합니다.

Step 5 – 렌더 패스

Vulkan의 렌더 패스는 렌더링 연산 중 사용하게 될 이미지의 타입을 기술하고, 얼마나 사용될 지, 그리고 어떻게 다루어 져야 할 지 설명합니다. 우리의 삼각형 렌더링 응용 프로그램의 초반에는 Vulkan에게 단일 이미지를 생상 타겟으로써 이용하고, 그리기 연산 전에 단색으로 지울 것을 지정합니다. 렌더 패스는 이미지의 타입만 기술하게 되므로, VkFramebuffer는 실제로 특정 이미지를 이러한 슬롯들에 바인딩하게 될 것입니다.