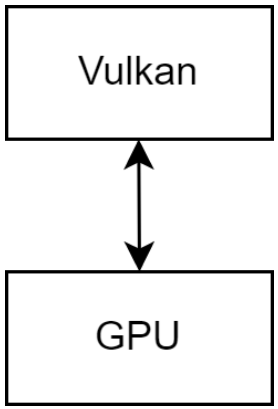
Esplorare L'API Grafica Vulkan

Emanuele Franchi

Vulkan

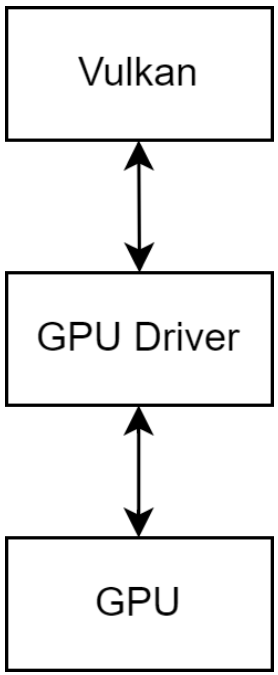
Vulkan Come API Grafica

- Vulkan è un'API grafica cross platform di nuova generazione
- Un'API grafica è un'interfaccia che ci permette di interagire con la GPU



Vulkan Come Specifica

- Vulkan è una specifica
- Non esiste un'unica implementazione
- Il gruppo Khronos specifica il comportamento dell'API
- I produttori di schede grafiche, se desiderano supportare Vulkan, devono fornirne un'implementazione
- Tale implementazione viene realizzata dal driver della scheda video



OpenGL

- Predecessore di Vulkan
- Anno di rilascio: 1992
- Sviluppata usando come modello l'architettura delle GPU dell'epoca
- Estesa in seguito per esporre le funzionalità delle nuove schede grafiche
- Questo ha portato ad una complessità crescente dei driver
- Questo causa overhead e inconsistenze tra driver diversi
- Sviluppata quando le CPU erano prevalentemente single core
- Sviluppata per essere usata in un contesto single threaded

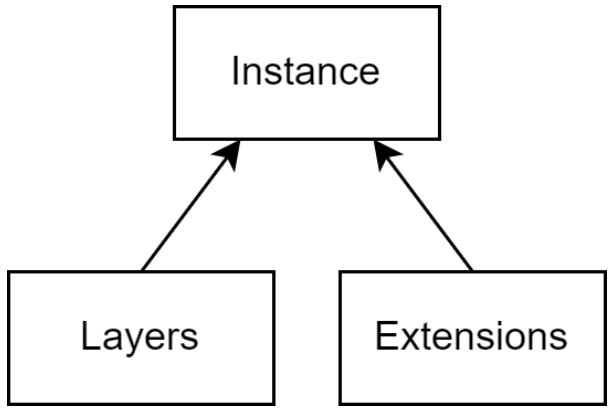
Vulkan

- Successore di OpenGL
- Anno di rilascio: 2016
- Sviluppata usando come modello l'architettura delle GPU odierne
- Driver molto più semplici e leggeri
- A più basso livello rispetto ad OpenGL
- Richiede più conoscenze da parte del programmatore
- Più verbosa rispetto ad OpenGL
- Può risultare in performance migliori, se il programmatore la usa coscientemente
- Sviluppata per essere usata in un contesto multithreaded
- Essendo così nuova, le GPU meno recenti non la supportano

Inizializzare Vulkan

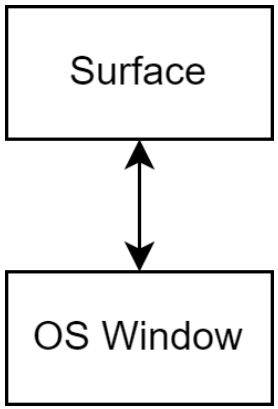
Vulkan Instance

- Dobbiamo creare un'istanza di Vulkan per accedere al resto dell'API
- Quando creiamo un'istanza indichiamo i layer che vogliamo abilitare
- I layer sono componenti opzionali che modificano il comportamento delle funzioni dell'API
- Per esempio, possiamo usare un layer per controllare se il nostro utilizzo dell'API non violi la specifica
- Quando creiamo un'istanza indichiamo le estensioni d'istanza che vogliamo abilitare
- Le estensioni aggiungono nuove funzionalità all'API



Finestra e Vulkan Presentation Surface

- Creiamo una finestra usando l'API del sistema operativo
- Creiamo una Vulkan presentation surface per interagire con la finestra creata

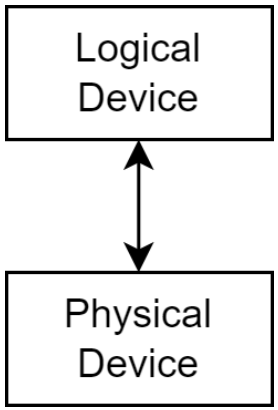


Selezionare Un Device Fisico

- Dobbiamo selezionare la GPU che andremo a utilizzare
- Deve supportare operazioni grafiche, quindi deve avere almeno una coda che possa eseguire tali comandi
- Deve supportare operazioni di presentazione di immagini, quindi deve avere almeno una coda che possa eseguire tali comandi
- Deve supportare la creazione di una swapchain
- Deve supportare almeno una modalità di presentazione compatibile con la presentation surface che abbiamo creato precedentemente

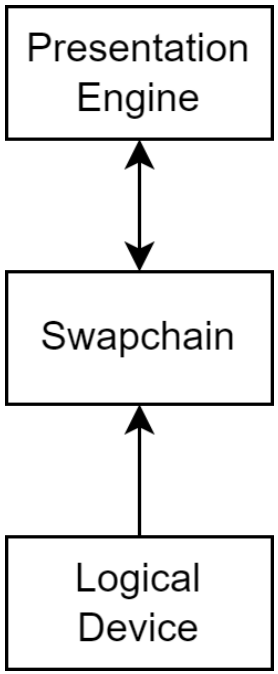
Creare Un Device Logico

- Per interagire con il device fisico selezionato, dobbiamo creare un device logico
- Quando creiamo il device logico, indichiamo la creazione di una coda per eseguire comandi grafici, e una coda per eseguire comandi di presentazione
- Una volta creato il device logico, possiamo ottenere le code richieste
- Se una coda supporta sia operazioni grafiche che di presentazione, possiamo usarla singolarmente



Creare Una Swapchain

- Una volta creato un device logico, lo possiamo utilizzare per creare una swapchain
- Dobbiamo creare una swapchain per presentare immagini sulla presentation surface
- Una swapchain crea e gestisce le immagini che saranno presentate
- Tramite la swapchain dobbiamo specificare la modalità di presentazione che verrà usata dal presentation engine del sistema operativo



Renderizzare Un Colore A Schermo

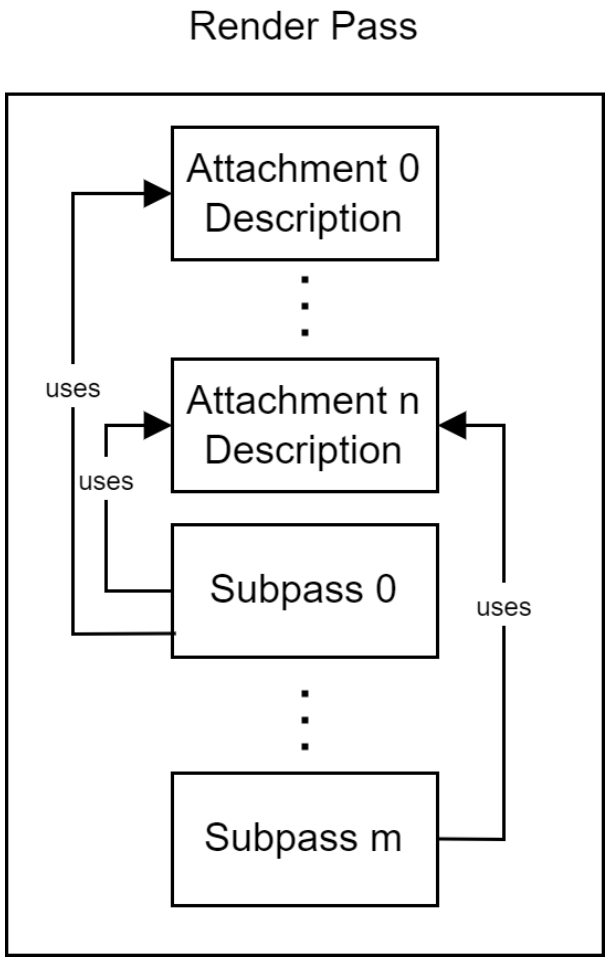
Risultato

Setup

- Creiamo due semafori: `imageAvailableSemaphore` e `renderFinishedSemaphore`
- Li useremo per sincronizzare il rendering e la presentazione
- Creiamo un command buffer su cui possiamo registrare dei comandi grafici
- Creiamo una fence che useremo assieme al command buffer durante il main loop

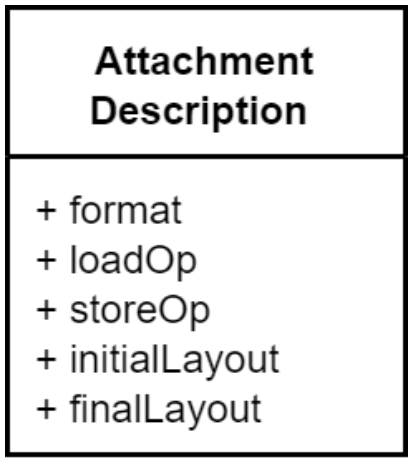
Render Pass

- Durante la fase di setup creiamo un render pass
- Un render pass ci permette di descrivere le immagini (attachment) che vengono utilizzate durante il rendering
- Un render pass suddivide le operazioni di rendering che utilizzano le stesse immagini in uno o più subpass



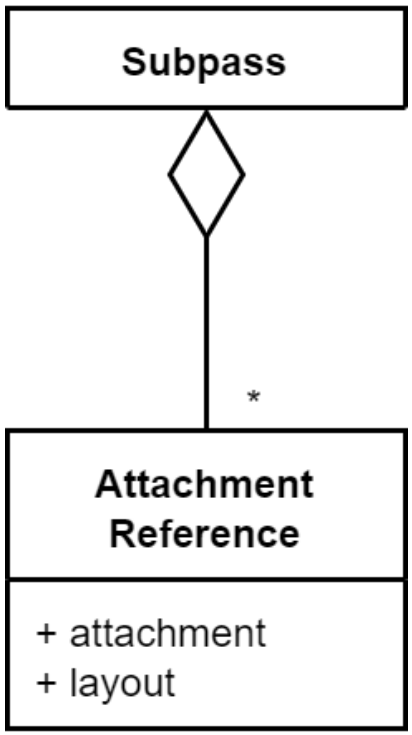
Attachmnet Description

- Abbiamo un solo attachment: una delle immagini della swapchain
- Vogliamo pulire l'attachment prima di utilizzarlo per la prima volta
- Volgiamo mantenere il contenuto dell'attachment quando il render pass termina
- Non ci importa del layout dell'attachment prima di iniziare il render pass
- Quando il render pass termina, volgiamo passare a un layout che renda ottimale la presentazione dell'attachment



Subpass

- Abbiamo un solo subpass che usa il nostro attachment
- Durante questo subpass usiamo l'attachment come color target
- Per questo motivo, durante questo subpass, il nostro attachment deve avere un layout adeguato



Main Loop (1)

- Otteniamo un'immagine dalla swapchain
- Non è garantito che questa immagine possa essere immediatamente utilizzata
- `imageAvailableSemaphore` viene segnalato quando l'immagine è disponibile
- Aspettiamo che la GPU termini l'esecuzione dei comandi precedenti
- Facciamo questo, aspettando sulla fence che abbiamo creato
- Creiamo un framebuffer per il nostro render pass, che contiene l'immagine della swapchain che abbiamo ottenuto
- Registriamo due comandi sul command buffer: uno per iniziare un'istanza del nostro render pass, e uno per terminarla
- Iniziando il render pass, specifichiamo il colore da scrivere nel color target

Main Loop (2)

- Inviamo il command buffer alla GPU
- La GPU aspetta su `imageAvailableSemaphore` prima di iniziare l'esecuzione dei comandi
- La GPU segnala `renderFinishedSemaphore` quando l'esecuzione dei comandi è terminata
- Inviamo un comando di presentazione alla GPU (non ci serve un command buffer)
- Specifichiamo l'immagine su cui abbiamo renderizzato il colore
- La GPU aspetta su `renderFinishedSemaphore` prima di iniziare la presentazione