Architektur und Programmierung von Grafik- und Koprozessoren

Programmieren mit dem Vulkan API

Stefan Zellmann

Lehrstuhl für Informatik, Universität zu Köln

SS2018

Lernziele

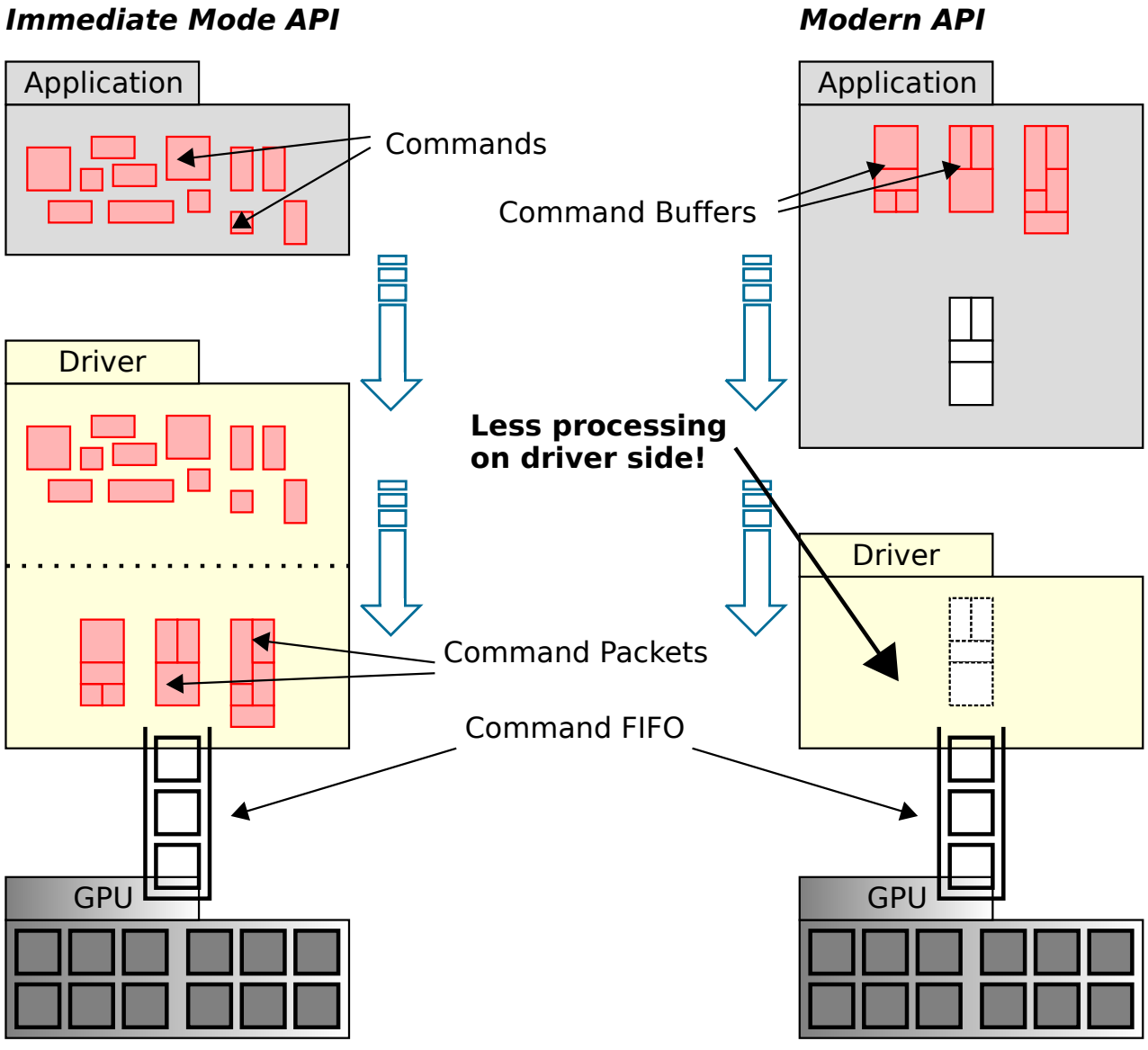
1. **Moderne Grafik APIs** - die Studenten lernen ein modernes Grafik API kennen, das die vorher gelernten Konzepte bzgl. der Grafik Pipeline abbildet.
2. **Barriers, Queues und Buffers** - die Studenten lernen Programmierkonzepte kennen, durch die moderne Grafik APIs Nebenläufigkeit auf Host-Ebene erzielen.
3. **Rendering Applikation** - die Studenten können eine einfache Applikation mit Vulkan entwickeln, die Dreiecksnetze anzeigen kann.

Überblick

Überblick

- Khronos Group Konsortium - `https://www.khronos.org/vulkan/`
- API 2015 vorgestellt. Alternative zu OpenGL.
- Ein API für Desktop und Mobile.
- Architekturdetails werden nicht versteckt (alte APIs), sondern *exponiert*.
  - Weniger CPU Overhead.
  - Weniger Treiberkomplexität.
  - Bessere Unterstützung für CPU Nebenläufigkeit.

Geringerer Treiber Overhead



Geringerer Treiber Overhead

- Immediate Mode APIs (OpenGL): GPU Kommandos werden immer wieder submittiert. *Treiber* implementiert Heuristiken, um Kommandos zu bündeln. Kein Applikationsspezifisches Wissen.
- Moderne APIs (Vulkan, D3D12): Applikation verwaltet Command Buffers und submittiert in Command Queues.
  - Abfolgen von Kommandos werden “recorded” und können so beliebig oft “ abgespielt” werden.
  - Applikationsspezifisches Wissen - Render Passes, Synchronisation, etc.
  - Kein Error Checking durch Treiber, mehr Verantwortung auf Applikationsseite ($\Rightarrow$ höheres Fehlerpotential).

Vulkan vs. Traditionelle APIs

Sowohl Vulkan als auch OpenGL werden von Khronos Group spezifiziert (in Teilen auch entwickelt). Vulkan ist OpenGL Nachfolger, ersetzt es jedoch nicht. OpenGL wird laut Khronos Group noch lange Bedeutung haben.

Frage daher: wann Vulkan, wann OpenGL? Applikationen, die

- CPU seitig multithreaded sind und/oder
- gleichzeitig mehrere GPUs nutzen und/oder
- cross-platform auf Mobiltelefonen und Desktops laufen und/oder
- CPU-bound sind,

profitieren vom Vulkan API (Laufzeit, Produktivität). In anderen Fällen OpenGL ggf. produktiver.

Vulkan Programmiermodell

Layer Modell

Loader als Schnittstelle zwischen Applikation und Treiber

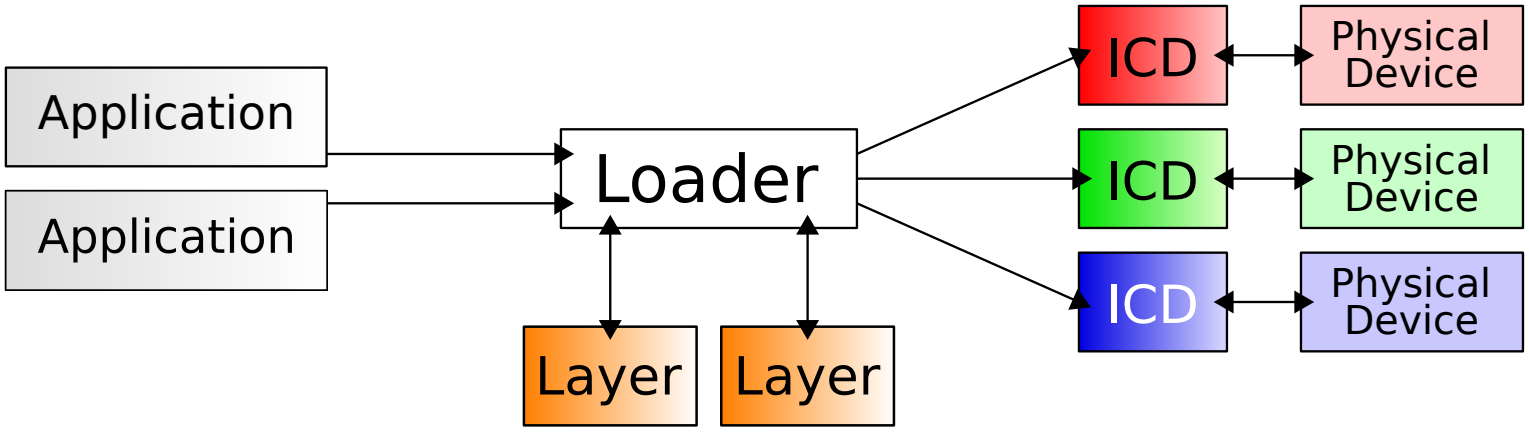


Abbildung: vgl. Khronos Spec.: Architecture of the Vulkan Loader Interface.

Layer Modell

Loader als Schnittstelle zwischen Applikation und Treiber

- Andere APIs: *Runtime*, die API Calls direkt an Treiber weiterleitet.
- Vulkan: Applikation linked mit *Loader* (`vulkan-1.dll`, `libvulkan.so.1`).
- Anwendung kommuniziert nicht direkt mit Treiber (Vulkan: *installable client driver* (ICD)), sondern mit Loader.
- Loader kann bestimmte Logik selbst abbilden (Windows Registry, JSON Manifest Files etc.), ohne mit Treiber zu kommunizieren oder in Kernel Mode zu wechseln.

Layer Modell

Loader als Schnittstelle zwischen Applikation und Treiber

- Loader verwaltet Extensions. Anders als z. B. OpenGL - Loader kann Extensions enumerieren, ohne sie tatsächlich zu laden (Windows Registry, Manifest Files).
- Vulkan erlaubt mehrere Treiber gleichzeitig, werden vom Loader verwaltet.
  - So können aus einer Vulkan Applikation mehrere Vulkan Devices genutzt werden, z. B. gleichzeitig integrierte Intel Grafik und *dedizierte* Nvidia/AMD GPU.
  - Szenario mit traditionellen APIs schwer abzubilden.
- Loader kann funktionale Layers zwischen Applikation und ICDs einfügen.

Vulkan Dokumentation

1.) man Pages:
`https://www.khronos.org/registry/vulkan/specs/1.1-extensions/man/html/`

2.) Spezifikation:
`https://www.khronos.org/registry/vulkan/specs/1.1-extensions/html/vkspec.html`

3.) Vulkan Tutorial (unabhängig, Alexander Overvoorde):
`https://vulkan-tutorial.com/`

4.) Intel: API without secrets (unter
`https://software.intel.com/`

5.) Lunar-SDK Doku (SDK, s. u.):
`https://vulkan.lunarg.com/doc/view/1.0.37.0/linux/vkspec.chunked/index.html`

Wesentliche Teile dieses Vorlesungsteils basieren auf den Tutorials von Intel und von Alexander Overvoordes.

Vulkan SDK

- LunarG (ehemals Valve) entwickelt Vendor-übergreifendes SDK für Windows, Linux und MacOS: `https://www.lunarg.com/vulkan-sdk/`
- Platform SDKs + Reihe von Layers.

Vulkan SDK

Wir lernen das gebräuchlichere ANSI-C API (`#include <vulkan/vulkan.h>`).

Es gibt auch ein C++ API (`#include <vulkan/vulkan.hpp>`). Bei Produktiveinsatz ggf. empfehlenswerter, jedoch weniger gebräuchlich. Bringt diverse Vereinfachungen z. B. durch RAII Klassen mit.