VULKAN®

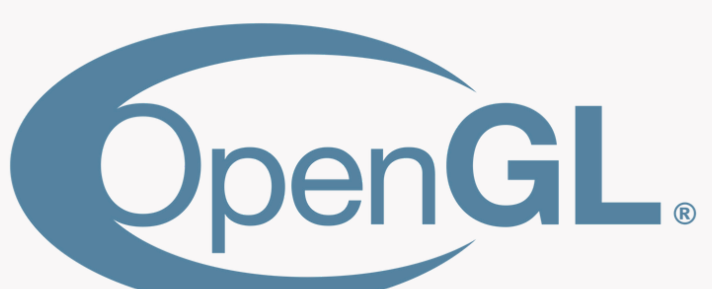
Computação Gráfica - IBT0058

Alunos: Claudio Evangelista; Danilo Dutra;
Gustavo Borges; Matheus Araujo.

BIBLIOTECAS GRÁFICAS

Bibliotecas gráficas são conjuntos de funções, ferramentas e interfaces que facilitam a criação e manipulação de gráficos em aplicações de software

+portabilidade +eficiencia +facilidade



Vulkan é uma nova geração de API gráficas que fornece acesso multiplataforma a GPUs modernas

- O Vulkan foi anunciado pela **Khronos Group** em 2015'
- "~~Uma iniciativa de nova geração ao OpenGL~~"
- Foi baseado na API Mantle da AMD
- Código aberto

Uma ferramenta para desenvolvedores criarem aplicações com aceleração por hardware

Busca ser um novo padrão para industria gráfica

INTRODUÇÃO E FUNDAMENTOS

Com o **Vulkan**, desenvolvedores que trabalham **aplicações 3D** de **tempo real** de **alto desempenho** podem ter mais controle sobre GPUs e CPUs

Vulkan ofere uma API baixo nível para as aplicações, focando em hardware mais moderno

+ controle GPU

- uso de CPU

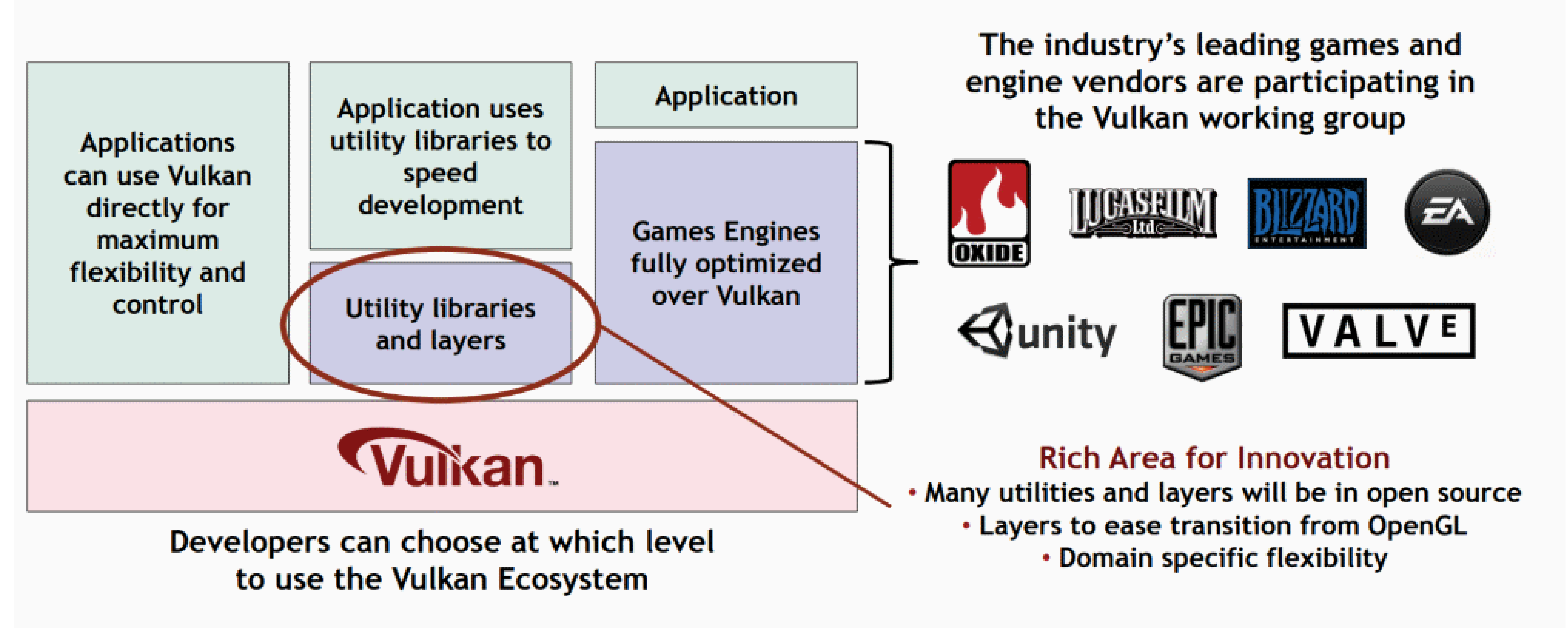
Projetado para multicore

INTRODUÇÃO E FUNDAMENTOS

Em seu núcleo, o Vulkan é uma especificação de API que é implantada em diferentes sistemas

API única em várias plataformas, com Windows, Linux, Mac, Android etc

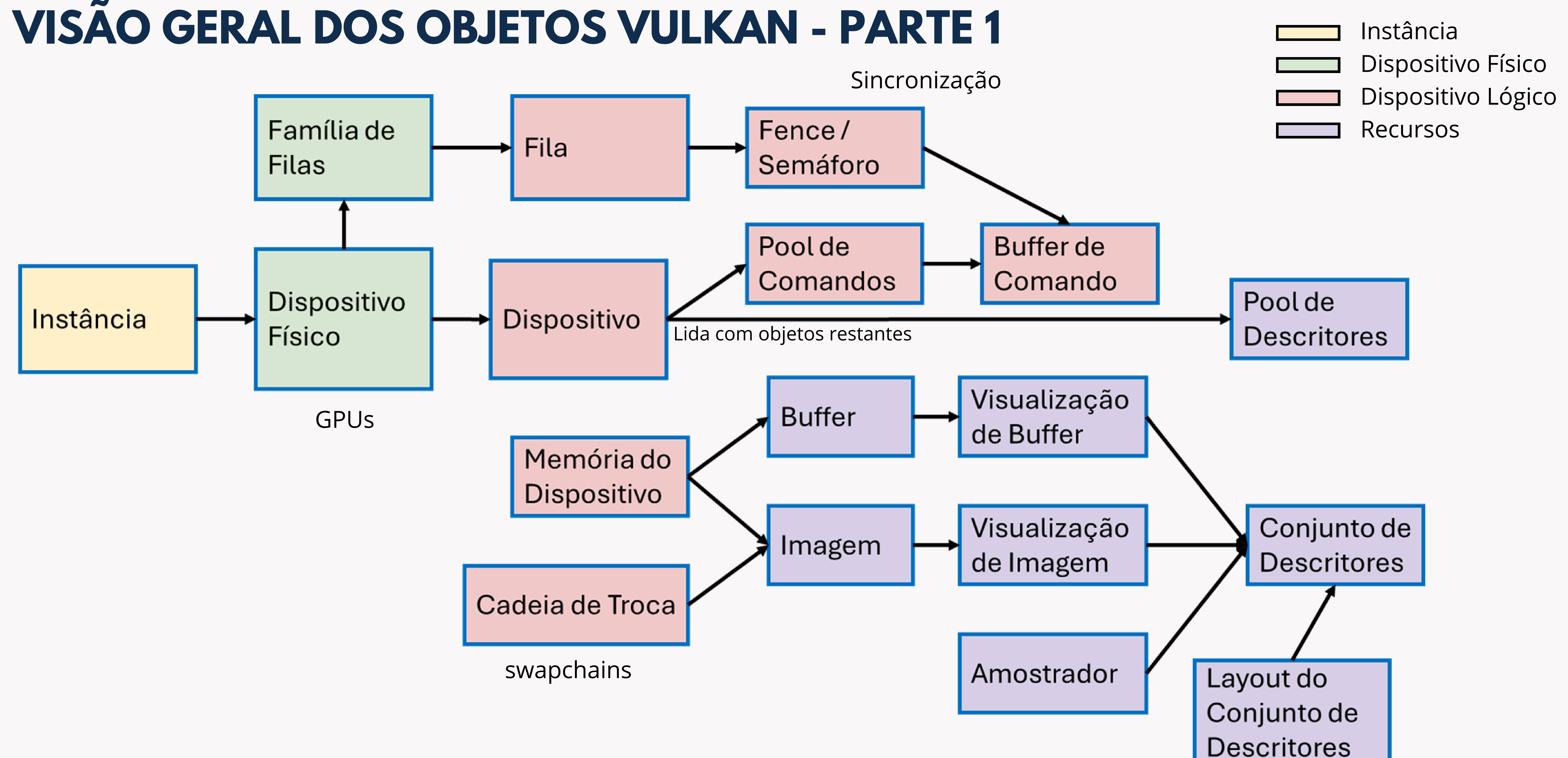
INTRODUÇÃO E FUNDAMENTOS



Abordarei os seguintes tópicos:

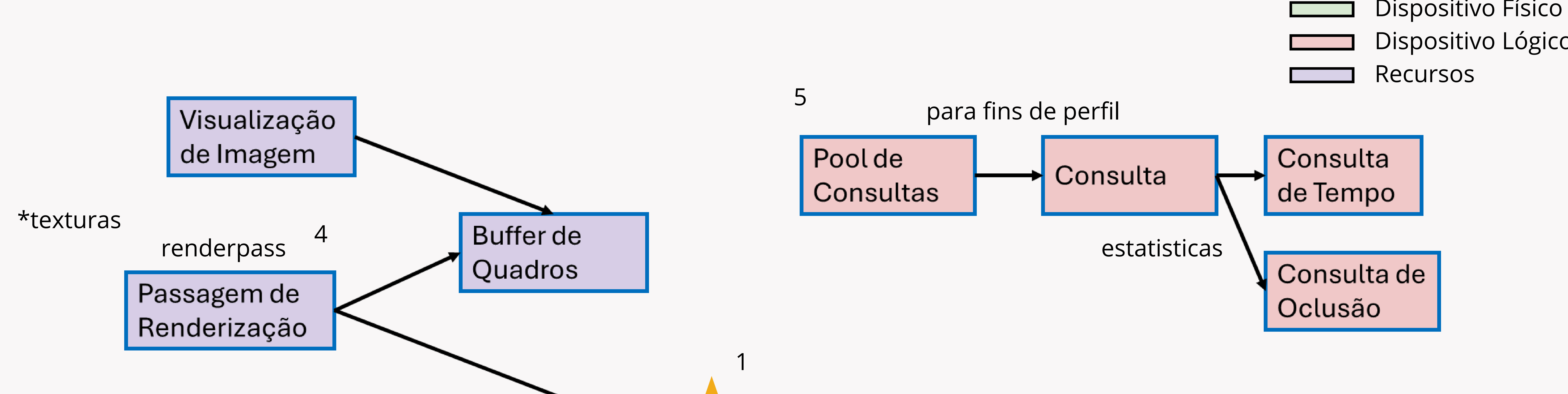
- Visão geral dos objetos vulkan; //resumo das mais importantes
- Dispositivos e Filas/Queues;
- Memoria (uso da VMA);
- ~~Recursos;~~
- Pipelines estruturas;
- ~~Render Pass;~~
- ~~Pipeline Layout;~~
- ~~Shader Module;~~
- ~~Compute Pipeline;~~
- ~~Graphics Pipeline;~~
- ~~Input Assembly;~~
- ~~Outros recursos...;~~
- Biblioteca wrapper V-EZ Vulkan;

VISÃO GERAL DOS OBJETOS VULKAN - PARTE 1



Continuar →

VISÃO GERAL DOS OBJETOS VULKAN - PARTE 2



1

3

Layout do Conjunto de Descritores

Layout do Pipeline

recursos

Pipeline

Descreve dados usados pela GPU para desenhar/executar shaders.

Cache de Pipeline

Módulo Shader

shader module

2

DISPOSITIVOS E FILAS

Contexto

VkInstance

```
VkResult vkCreateInstance( ... )

struct VkInstanceCreateInfo {
    VkInstanceCreateFlags      flags;
    const VkApplicationInfo*   pApplicationInfo;
    uint32_t                   enabledLayerCount;
    const char* const*         ppEnabledLayerNames;
    uint32_t                   enabledExtensionCount;
    const char* const*         ppEnabledExtensionNames;
}
```

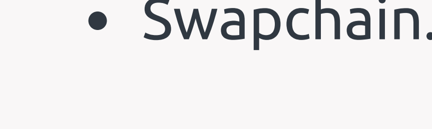
versao do vulkan

Camadas de instancia

Extensões

Exemplo:

- Surface;
- Swapchain.



DISPOSITIVOS E FILAS

*OpenGL consultar erros... criação de uma opção de registrar um callback
...é acionado quando um erro é encontrado.

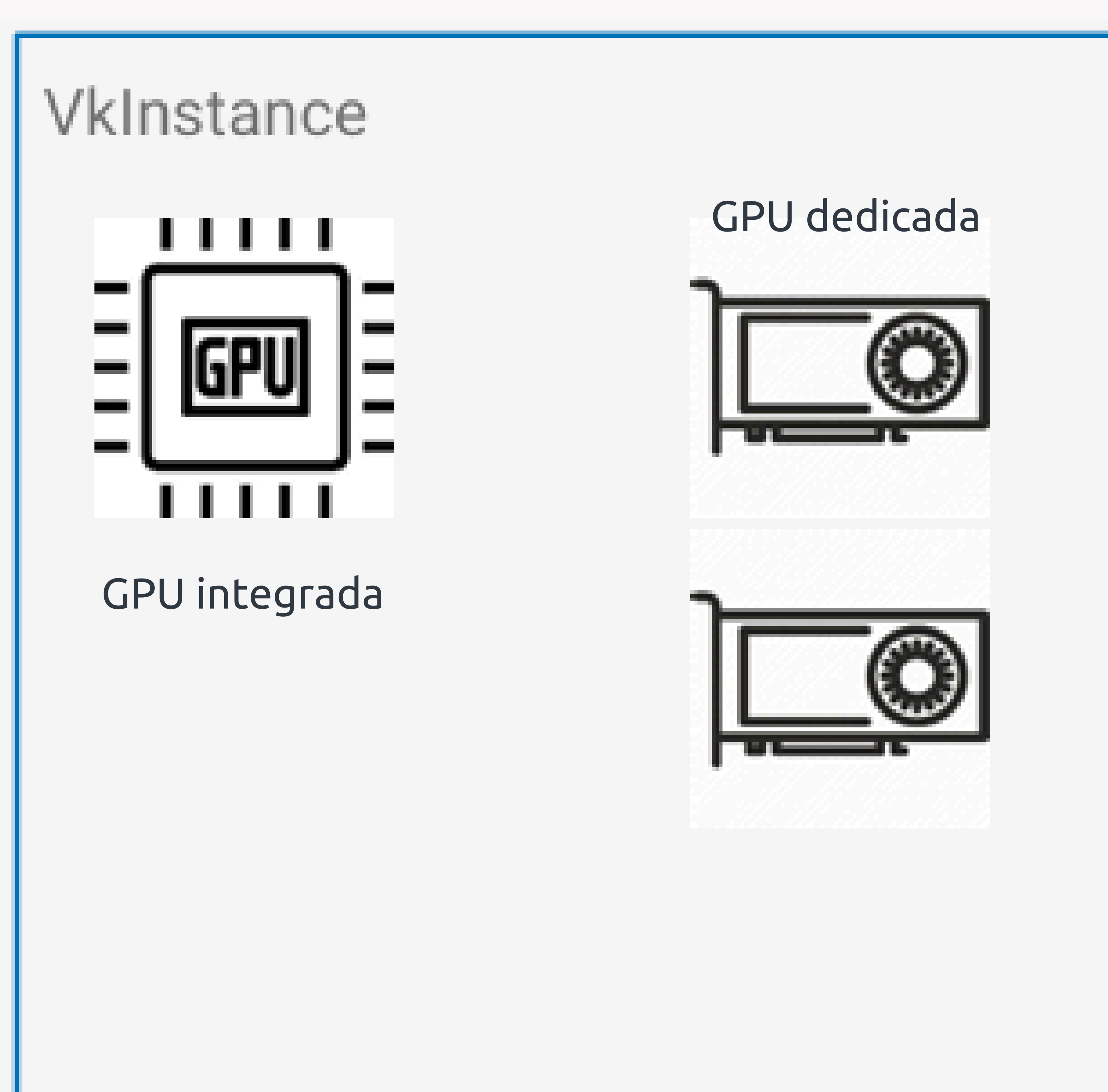
VkInstance

```
VkResult vkCreateInstance( ... )

struct VkInstanceCreateInfo {
    VkInstanceCreateFlags       flags;
    const VkApplicationInfo*    pApplicationInfo;
    uint32_t                    enabledLayerCount;
    const char* const*          ppEnabledLayerNames;
    uint32_t                    enabledExtensionCount;
    const char* const*          ppEnabledExtensionNames;
}

VK_LAYER_KHRONOS_validation
```

camada de validação
- Sempre será notificado mesmo que não registre um callback
- Mensagens de erros mais descritivas
- Recomendado sempre manter ativado

SDK opcional
Facilita o processo em controlar quais camadas estao habilitadas em tempo de execução
Permite adicionar ou remover camadas sem modificar o codigo

DISPOSITIVOS E FILAS




```
VkResult vkEnumeratePhysicalDevices( ... )

struct VkPhysicalDeviceProperties {
    uint32_t                            apiVersion;
    uint32_t                            driverVersion;
    uint32_t                            vendorID;
    uint32_t                            deviceID;
    VkPhysicalDeviceType                deviceType;
    char                                deviceName[];
    uint8_t                             pipelineCacheUUID[];
    VkPhysicalDeviceLimits              limits;
    VkPhysicalDeviceSparseProperties    sparseProperties;
}
```

* Existem APIs para consultar quais extensões e funcionalidades cada dispositivo expõe.

DISPOSITIVOS E FILAS

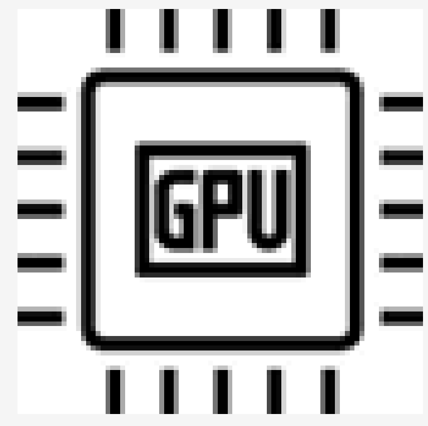
* Filas disponiveis para o dev usar p/ enviar comandos a GPU

Determinar quais filas

expõem

VkInstance

Dispositivo fisico



Podem expor diferentes tipos de filas, e para cada tipo, um dispositivo pode expor várias filas

```
VkResult vkGetPhysicalDeviceQueueFamilyProperties( ... )

enum VkQueueFlagBits {

    VK_QUEUE_GRAPHICS_BIT = 0x00000001,

    VK_QUEUE_COMPUTE_BIT = 0x00000002,

    VK_QUEUE_TRANSFER_BIT = 0x00000004,

    VK_QUEUE_SPARSE_BINDING_BIT = 0x00000008,

    VK_QUEUE_PROTECTED_BIT = 0x00000010,

    VK_QUEUE_VIDEO_DECODE_BIT_KHR = 0x00000020

}
```

Enviar trabalho à GPU

Realizar tarefas específicas, como renderização de gráficos ou processamento de dados.

Porque é importante conhecer os dispositivos fisicos (GPUs)?

- Conhecer os dispositivos físicos permite aproveitar todos os dispositivos disponíveis no seu sistema. Por exemplo, você poderia transferir parte do trabalho para a GPU integrada e depois passar o resultado para a GPU dedicada para processamento adicional.

- Tratamento de Múltiplas GPUs como um Único Dispositivo Lógico: Outra aplicação é tratar várias GPUs como um único dispositivo lógico. Isso pode ser útil para tarefas que se beneficiam do uso combinado de múltiplas GPUs.

* Diferentes dispositivos podem suportar diferentes extensões, e podemos controlar quais extensões são habilitadas para cada dispositivo.

Gerenciamento
OpenGL <- memoria <- driver

Dá para criar alocador de memória mas não é recomendado.
Library VMA (Vulkan Memory Allocator) -> Padrão em apps vulkan.

Host Memory

PCIE Bus

Device Memory

Não é acessivel diretamente ^-^

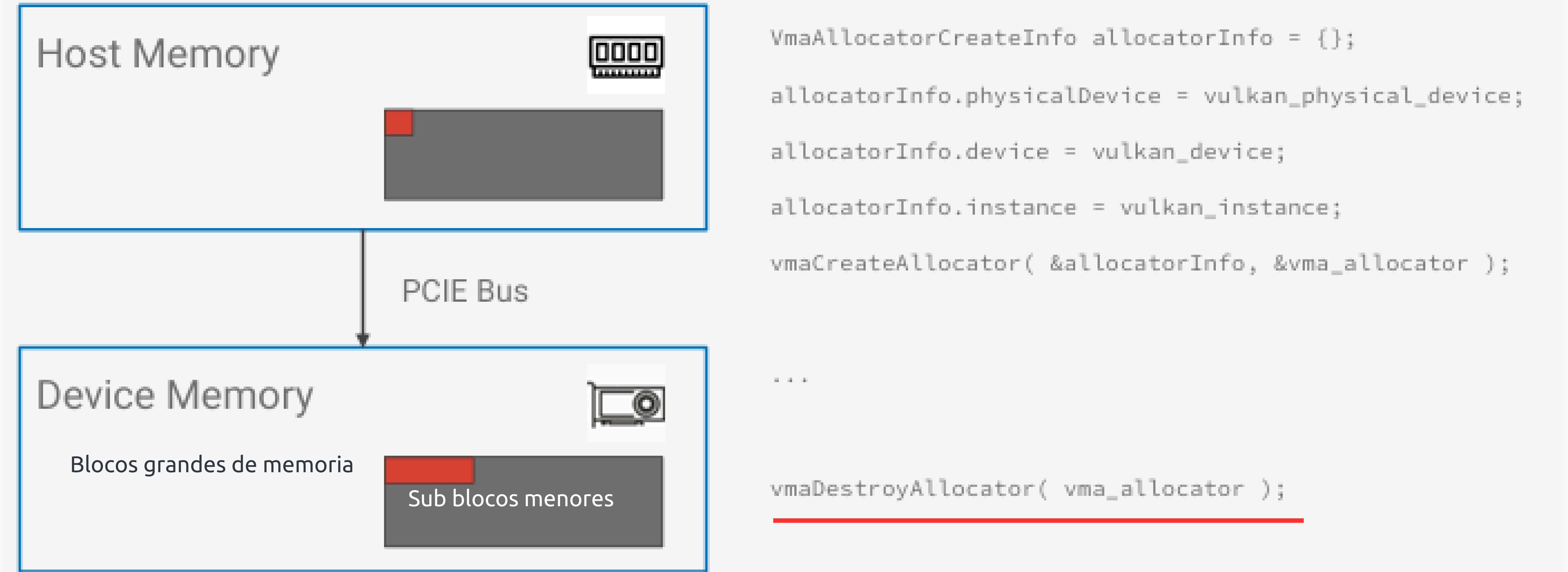
Da para acessar via recurso ou ReBAR (Resizable BAR)

```
enum VkMemoryPropertyFlagBits {
    VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT = 0x00000001,
    VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT = 0x00000002,
    VK_MEMORY_PROPERTY_HOST_COHERENT_BIT = 0x00000004,
    VK_MEMORY_PROPERTY_HOST_CACHED_BIT = 0x00000008,
    VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT = 0x00000010,
  // Provided by VK_VERSION_1_1
    VK_MEMORY_PROPERTY_PROTECTED_BIT = 0x00000020,
  // Provided by VK_AMD_device_coherent_memory
    VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD = 0x00000040,
  // Provided by VK_AMD_device_coherent_memory
    VK_MEMORY_PROPERTY_DEVICE_UNCACHED_BIT_AMD = 0x00000080,
  // Provided by VK_NV_external_memory_rdma
    VK_MEMORY_PROPERTY_RDMA_CAPABLE_BIT_NV = 0x00000100,
}
```

https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator

MEMÓRIA

Nos bastidores, o VMA cria multiplos pools de memoria dos quais ele faz sub-alocações.
Alocar memoria diretamente na GPU pode ser uma operação lenta.



Assim como com um pool de memória na CPU, isso também permite reutilizar a memória que foi liberada. O VMA também implementa recursos úteis, como rastreamento de recursos, para alertar você se alguns recursos não forem liberados quando a aplicação terminar.

https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator

MEMÓRIA

Importancia do VMA...

Criação de recursos

```
VkImage image;

vkCreateImage(device, pCreateInfo, nullptr, &image);

VkPhysicalDeviceMemoryProperties memoryProperties;

vkGetPhysicalDeviceMemoryProperties(physicalDevice,
&memoryProperties);

VkMemoryRequirements memoryRequirements;

vkGetImageMemoryRequirements(device, image,
&memoryRequirements);

findProperties(&memoryProperties,
memoryTypeBitsRequirement, requiredProperties);

VkDeviceMemory memory;

vkAllocateMemory(device, pAllocateInfo, nullptr,
&memory);

vkBindImageMemory(device, image, memory, memoryOffset);
```

Exemplos de criação de recursos (imagens e buffers)

Simplificação

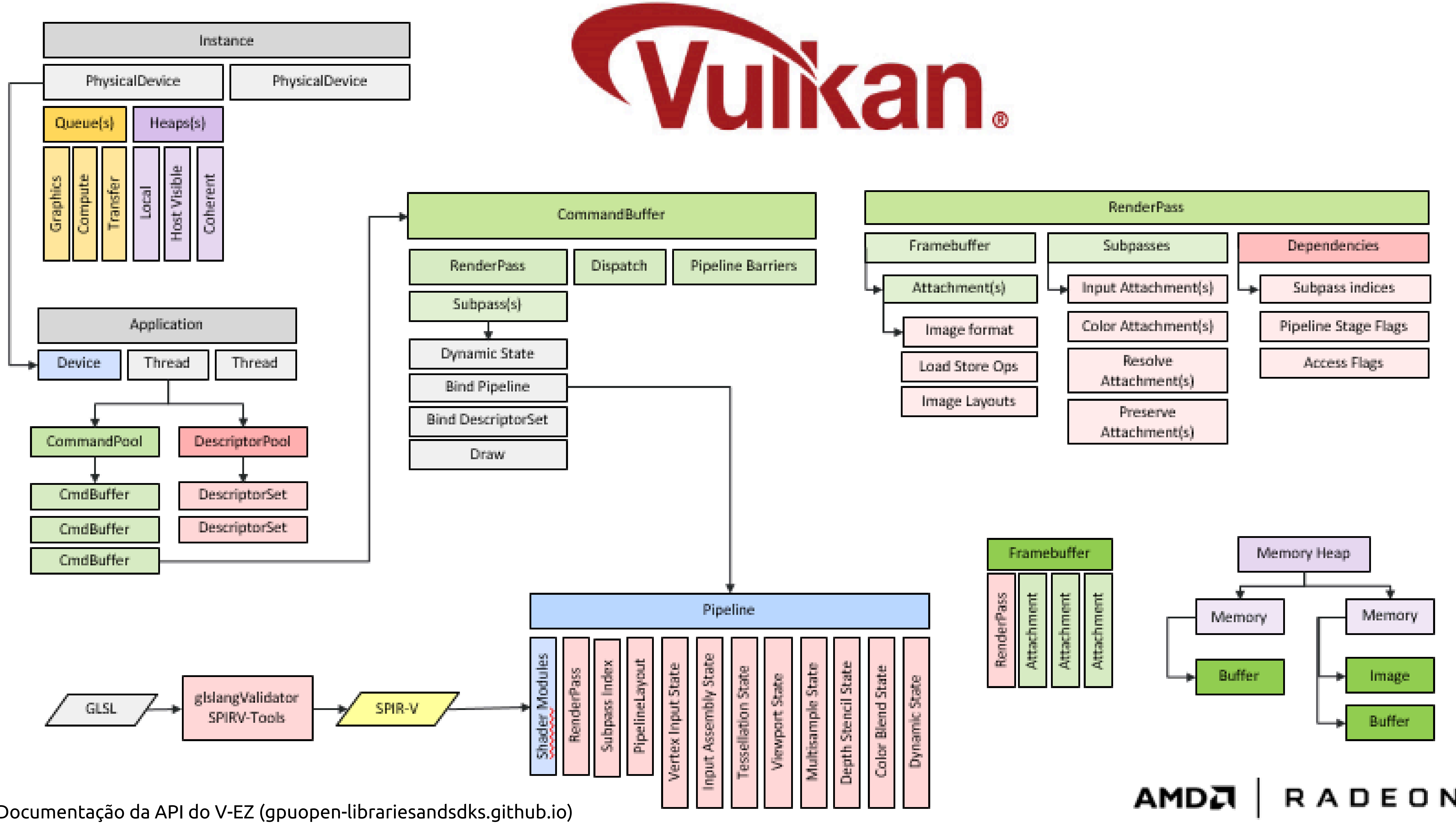
```
VmaAllocation vma_allocation;

VkImage image;

vmaCreateImage(vma_allocator, pCreateInfo, pMemoryInfo,
&image, &vma_allocation, nullptr);
```

PIPELINES

Input Assembler	Vertex Geometry Tessellation	Primitive Assembly	Rasterizer	Fragment	Output Merger

: : Tradução : :

Montador de entrada	Geometria de vértices Tesselação	Montagem primitiva	Rasterizador	Fragmento	Fusão de saída



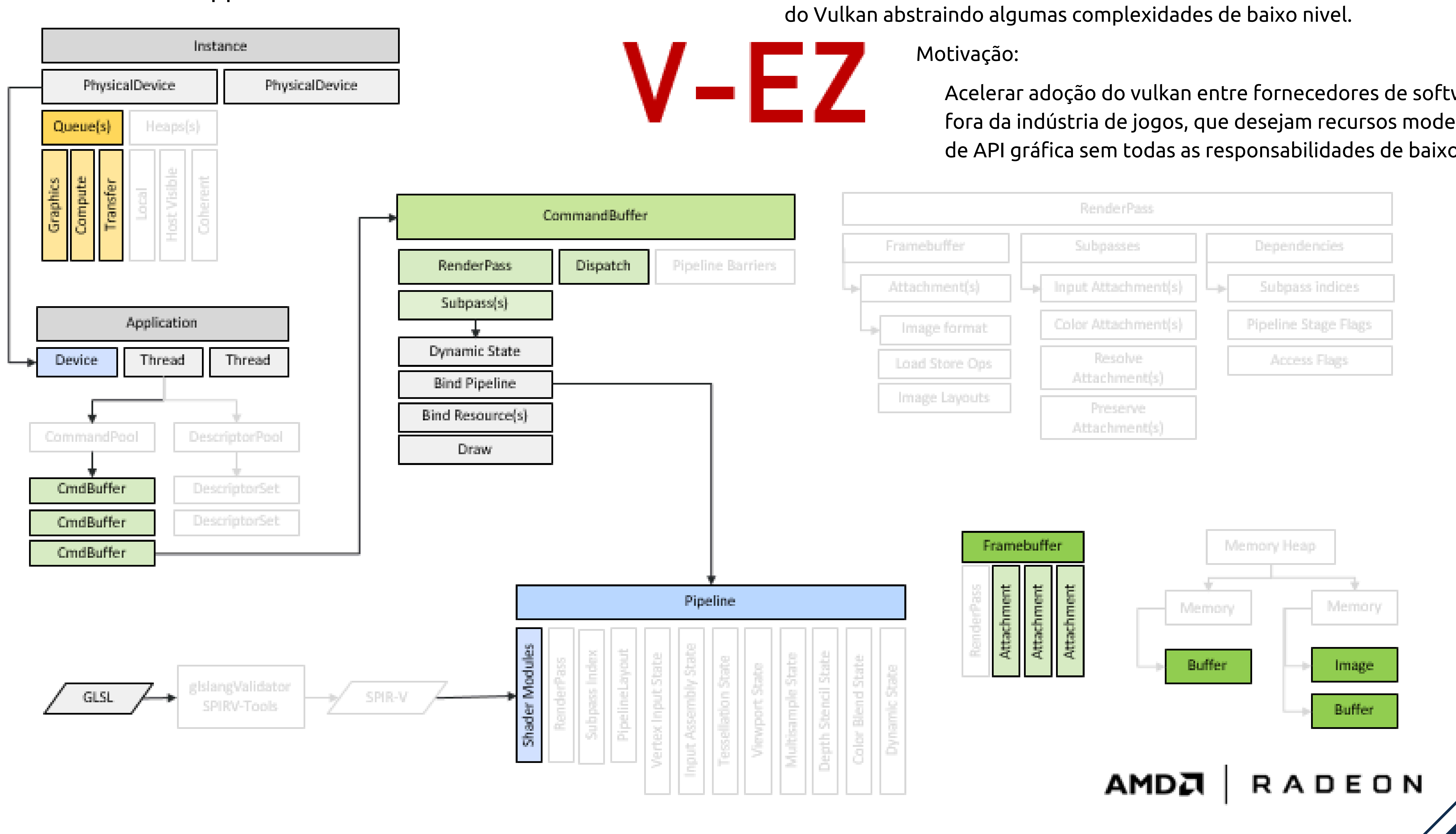
Documentação da API do V-EZ (gpuopen-librariesandsdks.github.io)

Biblioteca wrapper V-EZ Vulkan

Projetado para ser uma camada mais leve baseada em C em torno do Vulkan abstraindo algumas complexidades de baixo nivel.

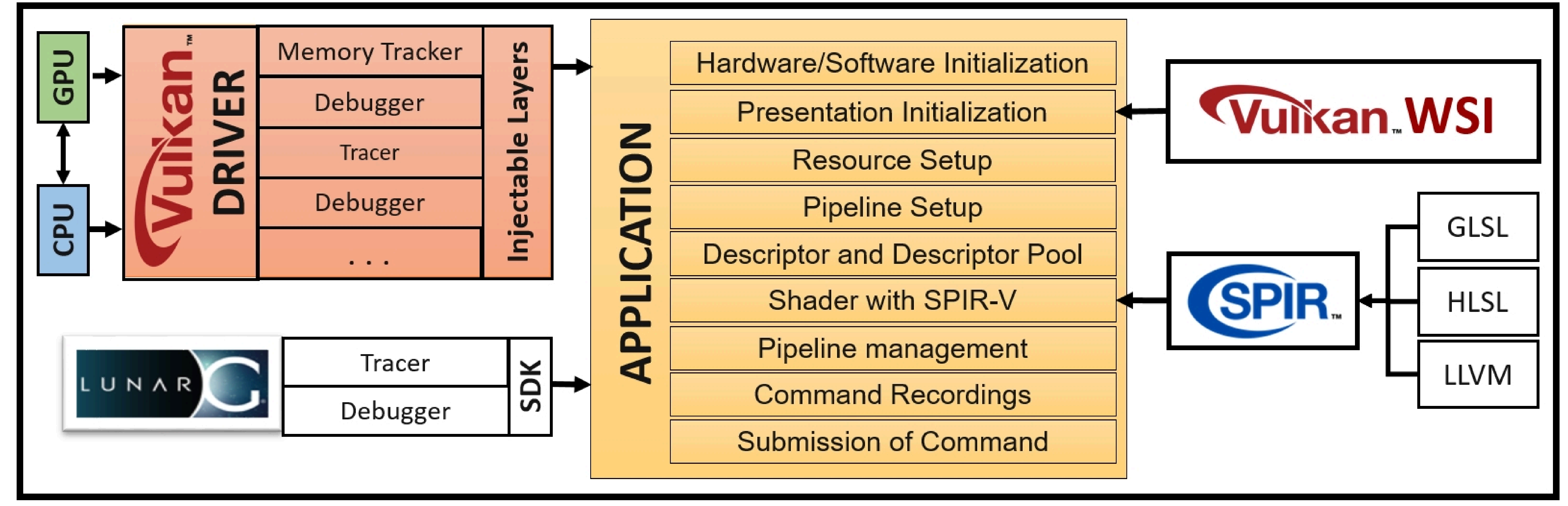
Motivação:

Acelerar adoção do vulkan entre fornecedores de software fora da indústria de jogos, que desejam recursos modernos de API gráfica sem todas as responsabilidades de baixo nível.



Documentação da API do V-EZ (gpuopen-librariesandsdks.github.io)

RESUMO GERAL



Understanding the Vulkan application - Learning Vulkan [Book] (oreilly.com)

+ BONUS

SPIR-V é uma representação intermediária binária usada para estágios de sombreamento gráfico e kernels de computação. No Vulkan, embora os sombreadores possam ser escritos em linguagens de alto nível como GLSL ou HLSL, é necessário gerar um binário SPIR-V ao usar vkCreateShaderModule. Ele oferece vantagens descritas em um white paper do Khronos e foi discutido em apresentações no Vulkan DevDay 2016.

APLICAÇÕES ATUAIS

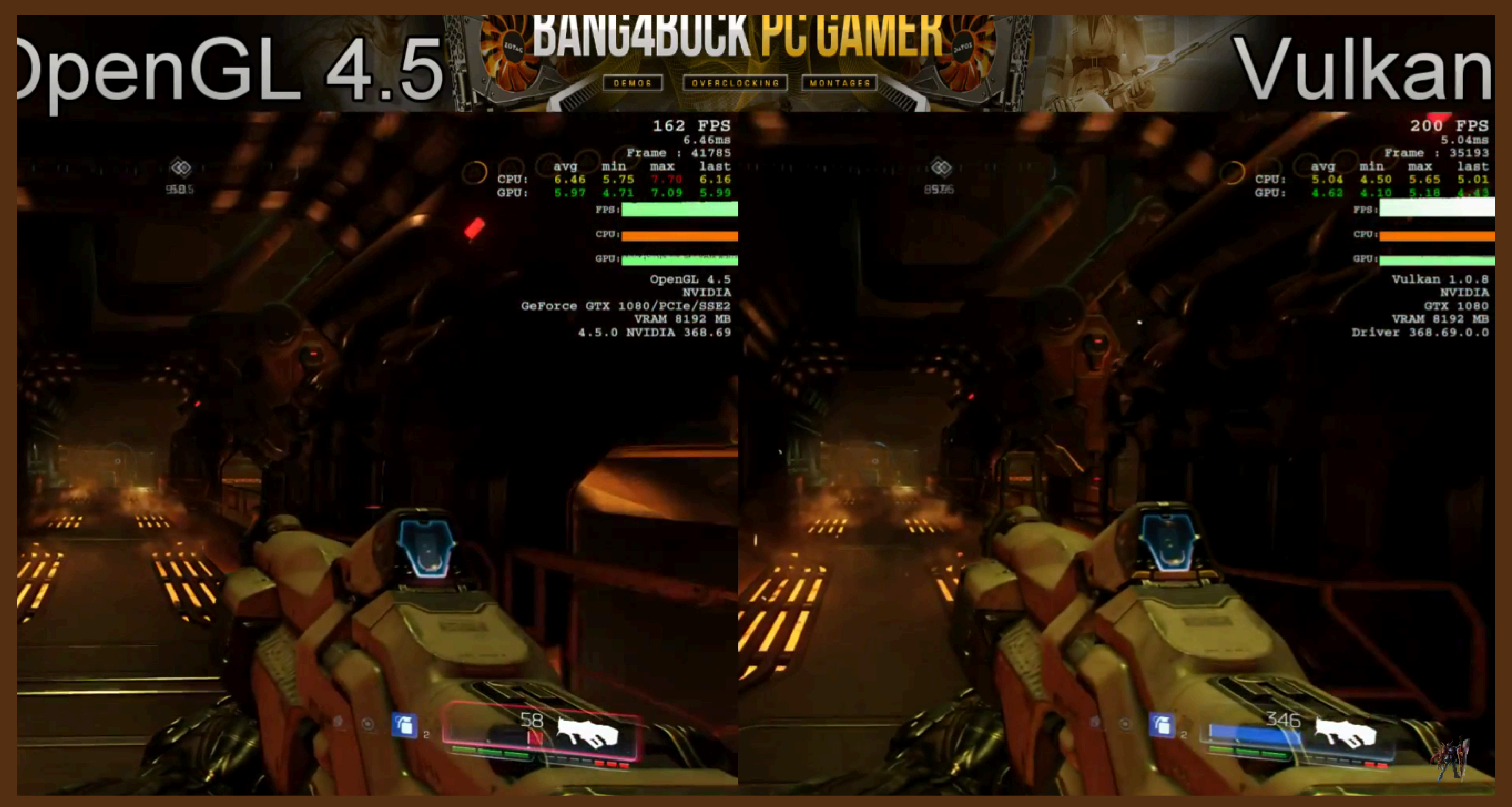
- renderizando doom(2016) com vulkan

-lançado com suporte a openGL e Vulkan

-desempenho com vulkan superior ao opengl

-mesmo utilizando rtx 1080, testes realizados mostrou um aumento de desempenho de cerca de 15% ao rodar o jogo com Vulkan em comparação com OpenGL

-slides posteriores explicando o alinhamento das gpus amd com o vulkan



OpenGL 4.5
BANG4BUCK PC GAMER
DEMOS
OVERCLOCKING
MONTAGES
Vulkan
162 FPS
6.46ms
Frame : 41785
avg min max last
CPU: 6.46 5.75 7.70 6.16
GPU: 5.97 4.71 7.09 5.99
FPS:
CPU:
GPU:
OpenGL 4.5
NVIDIA
GeForce GTX 1080/PCIe/SSE2
VRAM 8192 MB
4.5.0 NVIDIA 368.69
200 FPS
5.04ms
Frame : 35193
avg min max last
CPU: 5.04 4.50 5.65 5.01
GPU: 4.62 4.10 5.18
FPS:
CPU:
GPU:
Vulkan 1.0.8
NVIDIA
GTX 1080
VRAM 8192 MB
Driver 368.69.0.0
58
346

APLICAÇÕES

- ainda no que se refere ao desempenho do vulkan ao renderizar o doom, a mesma pode explicada com:

-menor overhead da cpu e eficiencia/aproveitamento multicore da gpu(paralelelismo)

essa otimização de hardware se dá pelo funcionemento do vulkan, o qual é composto por:

1)objetos vulkan(enfatizando, apenas):

controlar e organizar recursos gráficos em baixo nível

-buffer de vertices:

armazena dados de vertices

-pipeline grafico

configura como as operações gráficas sao processadas

APLICAÇÕES

- aplicações renderizadas em vulkan tendem a ser mais eficientes em gpus da amd

-vulkan foi originado de uma api da amd mais antiga chamada mantle

-herança de programação em baixo nivel e manipulação de recursos gráficos

-amd otimiza o suporte a programação em baixo nível e operações multithreading do vulkan

-arquitetura amd e drivers otimizados se alinha ao vulkan

arquiteturas RDNA e CDNA

APLICAÇÕES ATUAIS

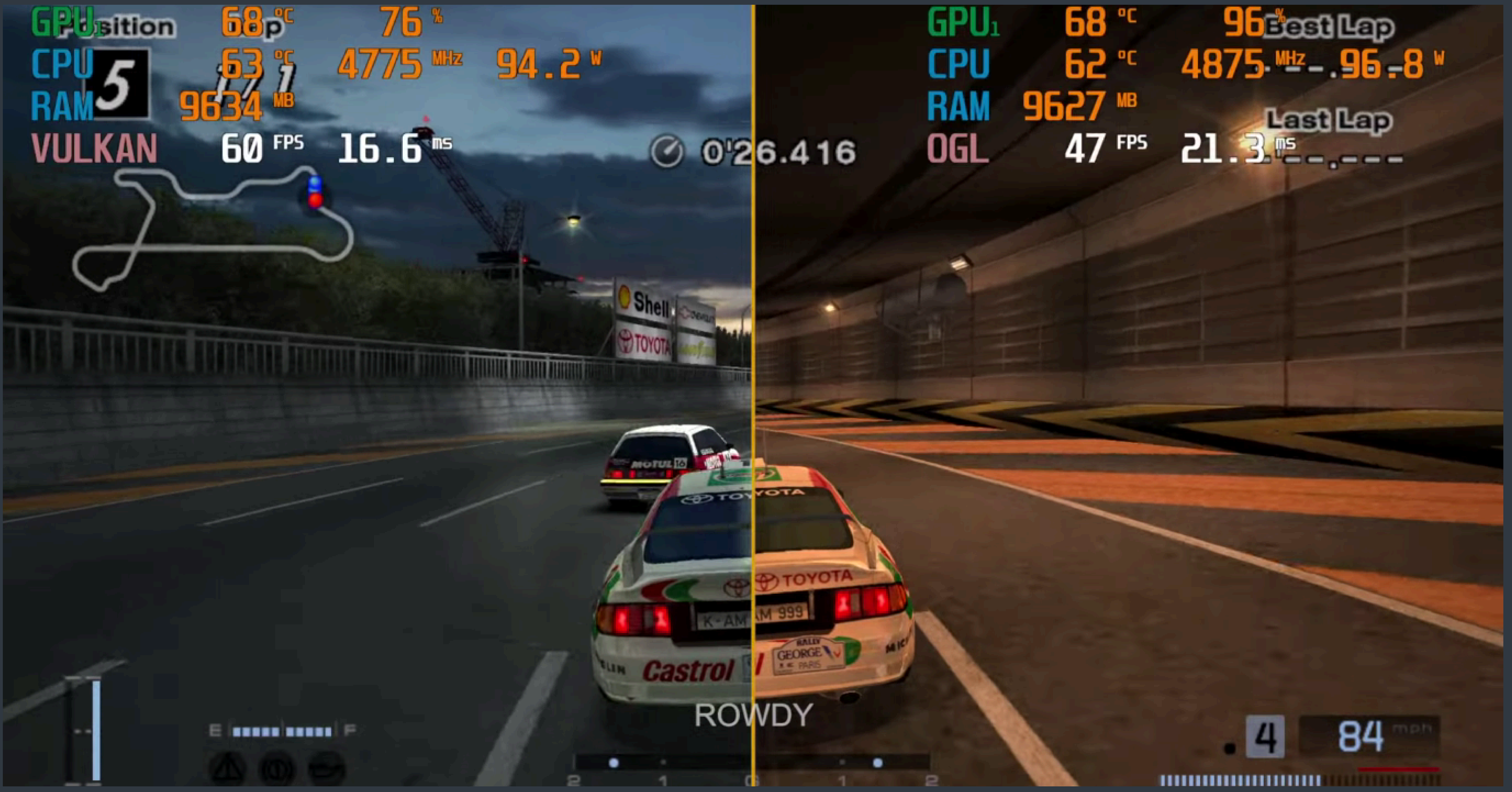
- alternativa ao direct3D e openGL no PCSX2, RPCS3, ETC

-melhora na emulação e desempenho em alguns jogos

- Para pcsx2, é uma alternativa ao openGL no linux

-era executado apenas com openGL

-suporte ao vulkan no pcsx2 trouxem todas as vantagens de uma api baixo-nivel



GPU 68 °C 76 %
CPU 63 °C 4775 MHz 94.2 W
RAM 9634 MB
VULKAN 60 FPS 16.6 ms
0'26.416
GPU1 68 °C 96 %
CPU 62 °C 4875 MHz 96.8 W
RAM 9627 MB
OGL 47 FPS 21.3 ms
Best Lap
Last Lap
ROWDY
4
84 mph

APLICAÇÕES ATUAIS

- fortnite mobile com vulkan

-em cenários mais limitados, o controle e manipulação do hardware é essencial

-melhor desempenho, eficiencia energética e qualidade gráfica

-manipulação em baixo nivel é essencial em dispositivos móveis

exemplo do fortnite mobile sendo renderizado pelo vulkan no celular OnePlus 8

APLICAÇÕES ATUAIS

- as vantagens de se utilizar vulkan nessas aplicações:

  -melhor desempenho

  -compatibilidade de plataforma

- desvantagens:

  -implementação complexa

  -pode nao ser compatível em dispositivos mais antigos, diferente do openGL

VULKAN X OPENGL

- ABSTRAÇÃO E COMPLEXIDADE
- DESEMPENHO E EFICIENCIA
- COMPATIBILIDADE E PORTABILIDADE
- PROJETOS E CASOS DE USO
- ECOSSISTEMA E SUPORTE

Referencias

[1]. Vulkan Documentation, (2024) "What is Vulkan?", https://docs.vulkan.org/guide/latest/what_is_vulkan.html, agosto.

[2]. Marco Castorina, Gabriel Sassone, (2023), "Vulkan for graphics research", https://diglib.eg.org/server/api/core/bitstreams/8d973d92-dedb-44e0-94bc-50f11cde9521/content.

[3]. OREILLY, (2024), "Understanding the Vulkan application", https://www.oreilly.com/library/view/learning-vulkan/9781786469809/ch01s05.html.

[4]. GPUOpen, Advanced Micro Devices, (2018), "V-EZ API Documentation", https://gpuopen-librariesandsdks.github.io/V-EZ/.

[5] ousadoBR (2024) "FORTNITE MOBILE ANDROID". YouTube. Disponível em: https://www.youtube.com/watch?v=GB3bmSTVZfI

[6] rowdy (2024) "Vulkan vs OpenGL Renderer Comparison". YouTube. Disponível em: https://www.youtube.com/watch?v=GB3bmSTVZfI

[7] Bang4BuckPC (2024) "Doom OpenGL VS Vulkan ". YouTube. Disponível em: https://www.youtube.com/watch?v=GB3bmSTVZfI

OBRIGADO!