

Vulkan API



Adam Sawicki „Reg”

<http://asawicki.info>

KNTG Polygon

18.04.2018



Agenda

- Co to jest Vulkan?
- Przegląd API
- Narzędzia
- Zalety i wady
- Czy jest dla mnie?

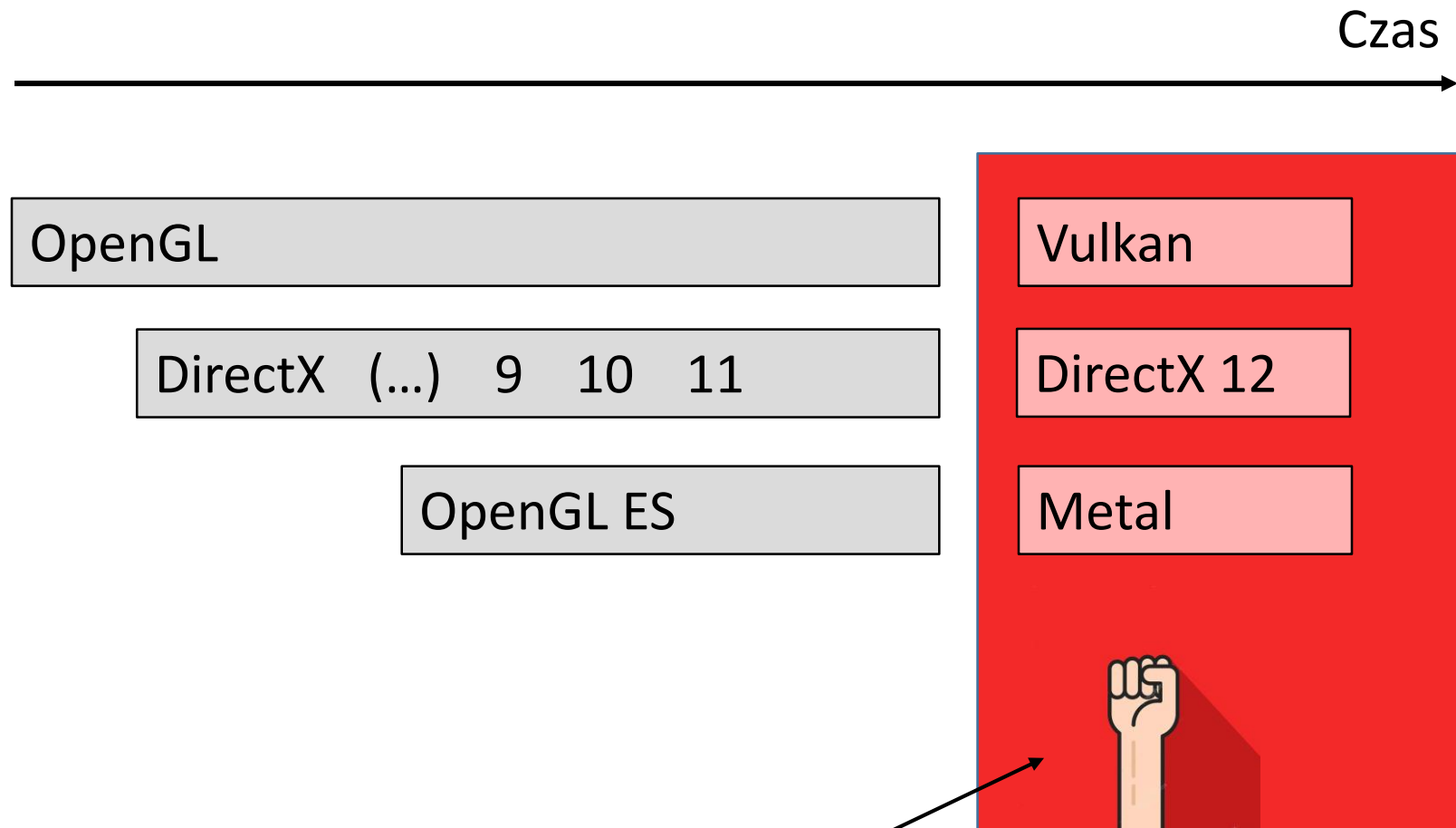
- Ogłoszenie 😊

Co to jest?

Vulkan to API graficzne



API graficzne – wg czasu



Rewolucja – API graficzne nowej generacji

API graficzne – wg platformy

- Vulkan stworzony przez **Khronos** jako otwarty standard
- Pierwsze w historii wspólne API na pecety i urządzenia mobilne

DirectX 12

Microsoft



Metal



Vulkan

KHRONOS
GROUP

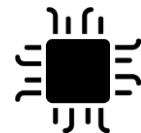


Jakie są te nowe API?

„Explicit API”

- Bez naleciałości historycznych
 - Lepiej odpowiadają budowie współczesnych GPU
- Niskopoziomowe
 - Blżej sprzętu
 - Bezpośrednia kontrola nad GPU
 - Prostszy sterownik
- Trudniejsze do opanowania i używania ☹️

~~glVertex()~~



Wrócimy do tego...

Jak zacząć?

- Ściągnij i zainstaluj **Vulkan SDK**
 - <https://www.lunarg.com/vulkan-sdk/>
- Czytaj specyfikację
 - <https://www.khronos.org/registry/vulkan/>
 - HTML (jeden wielki plik ☹) lub PDF

```
#include <vulkan/vulkan.h>
```

Zlinkuj vulkan-1.lib



Jak wygląda API?

- Zdefiniowane w **C** (enum, typedef, struct, funkcje globalne)
- Obiektowe

```
VkBufferCreateInfo bufCreateInfo = { VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO };  
bufCreateInfo.size = 65536;  
bufCreateInfo.usage = VK_BUFFER_USAGE_VERTEX_BUFFER_BIT |  
    VK_BUFFER_USAGE_TRANSFER_DST_BIT;  
  
VkBuffer buffer;  
VkResult res = vkCreateBuffer(device, &bufCreateInfo, nullptr, &buffer);  
  
// ...  
  
vkDestroyBuffer(device, buffer, nullptr);
```



Jak wygląda API?

Struktura opisująca tworzony bufor.
(Po utworzeniu parametrów już nie można zmienić.)



```
VkBufferCreateInfo bufCreateInfo = { VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO };
bufCreateInfo.size = 65536;
bufCreateInfo.usage = VK_BUFFER_USAGE_VERTEX_BUFFER_BIT |
    VK_BUFFER_USAGE_TRANSFER_DST_BIT;

VkBuffer buffer;
VkResult res = vkCreateBuffer(device, &bufCreateInfo, nullptr, &buffer);

// ...

vkDestroyBuffer(device, buffer, nullptr);
```

Jak wygląda API?

Obiekt (tak naprawdę wskaźnik)

```
VkBufferCreateInfo bufCreateInfo = { VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO };
bufCreateInfo.size = 65536;
bufCreateInfo.usage = VK_BUFFER_USAGE_VERTEX_BUFFER_BIT |
    VK_BUFFER_USAGE_TRANSFER_DST_BIT;
VkBuffer buffer;
VkResult res = vkCreateBuffer(device, &bufCreateInfo, nullptr, &buffer);

// ...

vkDestroyBuffer(device, buffer, nullptr);
```

Jak wygląda API?

```
VkBufferCreateInfo bufCreateInfo = { VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO };
bufCreateInfo.size = 65536;
bufCreateInfo.usage = VK_BUFFER_USAGE_VERTEX_BUFFER_BIT |
    VK_BUFFER_USAGE_TRANSFER_DST_BIT;

VkBuffer buffer;
VkResult res = vkCreateBuffer(device, &bufCreateInfo, nullptr, &buffer);

// ...

vkDestroyBuffer(device, buffer, nullptr);
```

Funkcje tworzące i niszczące obiekt

Jak wygląda API?

Powodzenie lub kod błędu:

VK_SUCCESS

VK_ERROR_OUT_OF_DEVICE_MEMORY

VK_ERROR_...

```
VkBufferCreateInfo bufCreateInfo = { VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO };
bufCreateInfo.size = 65536;
bufCreateInfo.usage = VK_BUFFER_USAGE_VERTEX_BUFFER_BIT |
    VK_BUFFER_USAGE_TRANSFER_DST_BIT;

VkBuffer buffer;
VkResult res = vkCreateBuffer(device, &bufCreateInfo, nullptr, &buffer);

// ...

vkDestroyBuffer(device, buffer, nullptr);
```


Budowa struktur

Prawie wszystkie struktury zaczynają się od 2 pól:

1. Enum identyfikuje typ struktury

Np. `VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO`

```
typedef struct VkBufferCreateInfo {  
    VkStructureType      sType;  
    const void*          pNext;  
    VkBufferCreateFlags  flags;  
    VkDeviceSize         size;  
    VkBufferUsageFlags   usage;  
    VkSharingMode        sharingMode;  
    uint32_t             queueFamilyIndexCount;  
    const uint32_t*      pQueueFamilyIndices;  
} VkBufferCreateInfo;
```




Budowa struktur

Prawie wszystkie struktury zaczynają się od 2 pól:

2. Wskaźnik na następną strukturę
Pozwala łączyć je w łańcuch z nowymi (rozszerzenia!)

```
typedef struct VkBufferCreateInfo {  
    VkStructureType      sType;  
    const void*          pNext;  
    VkBufferCreateFlags  flags;  
    VkDeviceSize         size;  
    VkBufferUsageFlags   usage;  
    VkSharingMode         sharingMode;  
    uint32_t             queueFamilyIndexCount;  
    const uint32_t*      pQueueFamilyIndices;  
} VkBufferCreateInfo;
```



Validation Layers

- W starych API każda funkcja sprawdza poprawność i zwraca rezultat

Crash oznacza błąd w sterowniku



- W Vulkanie wiele funkcji nie zwraca rezultatu
 - Funkcje tworzące zasoby, alokujące pamięć zwracają **VkResult**
 - Funkcje dodające komendy graficzne zwracają **void**

Validation Layers

- Nieprawidłowe użycie API powoduje niezdefiniowany skutek
 - Może zadziałać dobrze (na danym GPU i sterowniku)
 - Może zadziałać źle (nieprawidłowy wynik)
 - Może wywalić sterownik (crash, timeout „TDR”)
- W wychwyceniu błędów pomagają **Validation Layers**
 - Po włączeniu wykonują dodatkowe sprawdzenia, zgłaszają komunikaty błędów i ostrzeżeń
 - Kiedy wyłączone, nie dodają narzutu na CPU

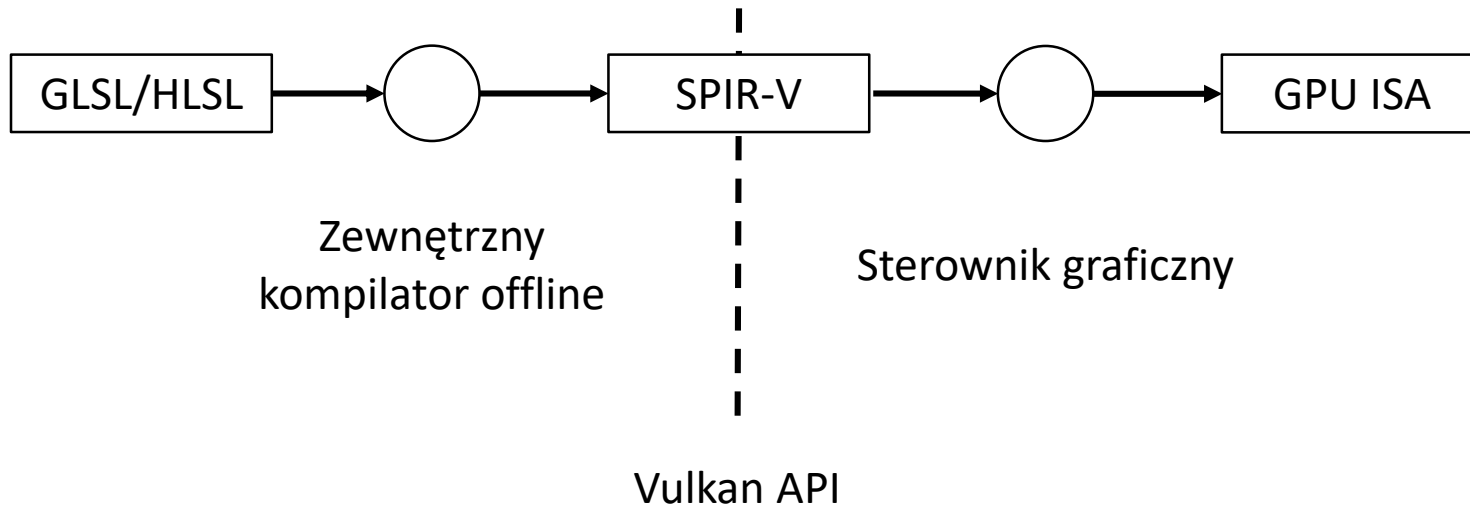


Podstawowe obiekty

- **VkInstance** – reprezentuje zainicjalizowany Vulkan Runtime
Utwórz tylko jeden na całą aplikację
- **VkPhysicalDevice** – reprezentuje GPU dostępne w systemie
Odpytaj VkInstance o ich listę, wybierz odpowiedni
- **VkDevice** – reprezentuje Vulkana zainicjalizowanego na danym GPU
Główny obiekt – jak ID3D11Device, OGL context

Shadery

- Obiekt **VkShaderModule**
- Vulkan przyjmuje shadery w formacie SPIR-V
Postać pośrednia, binarna, wzorowana na LLVM IR
- Język wysokiego poziomu poza Vulkanem
Dostępne kompilatory do **GLSL** i **HLSL**



Kolejki komend

Układy graficzne obsługują wiele kolejek

- Odpytaj VkDevice o obiekty **VkQueue**
- Dzielą się na 3 kategorie:
 - **Transfer** – wyłącznie kopiowanie danych i przesyłanie przez PCIe
 - **Compute** – compute shader
 - **Graphics** – pełny potok grafiki 3D
- Mogą działać równolegle
 - Async Compute – compute shader wykonywany równolegle z grafiką 3D
 - Transfer w tle, np. streamowanie tekstur



Kolejki komend

VkCommandBuffer – bufor poleceń do wykonania na GPU

- To do niego wpisujemy tradycyjne SetX(), SetY(), Draw()
- Można użyć wielu
 - Można je wypełniać równoległe na wielu wątkach
 - Można przygotować raz i wykonywać wiele razy

Kolejki komend

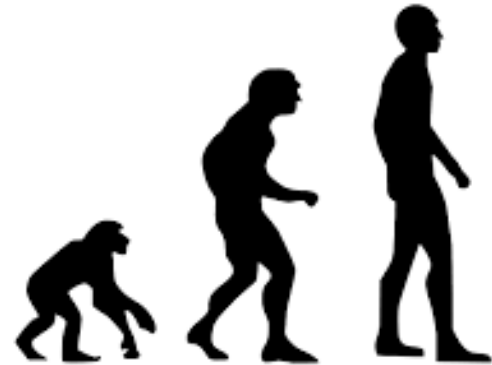


```
vkBeginCommandBuffer(cmdBuf, &cmdBufBeginInfo);  
vkCmdBeginRenderPass(cmdBuf, &renderPassBeginInfo,  
    VK_SUBPASS_CONTENTS_INLINE);  
vkCmdBindIndexBuffer(cmdBuf, indexBuffer, 0,  
    VK_INDEX_TYPE_UINT16);  
vkCmdDraw(cmdBuf, vertexCount, 1, 0, 0);  
vkCmdEndRenderPass(cmdBuf);  
vkEndCommandBuffer(cmdBuf);  
vkQueueSubmit(queue, 1, &submitInfo, nullptr);
```



Potok

- DirectX 9
 - Każdy stan ustawiany oddzielnie
 - SetRenderState, SetPixelShader, ...
- DirectX 11
 - Obiekty stanów, niezmiennie po utworzeniu
 - ID3D11BlendState, ID3D11DepthStencilState, ...
- DirectX 12, Vulkan
 - **VkPipeline** – jeden niezmienny obiekt z (prawie) całą konfiguracją potoku
 - Nazywany też Pipeline State Object (PSO)

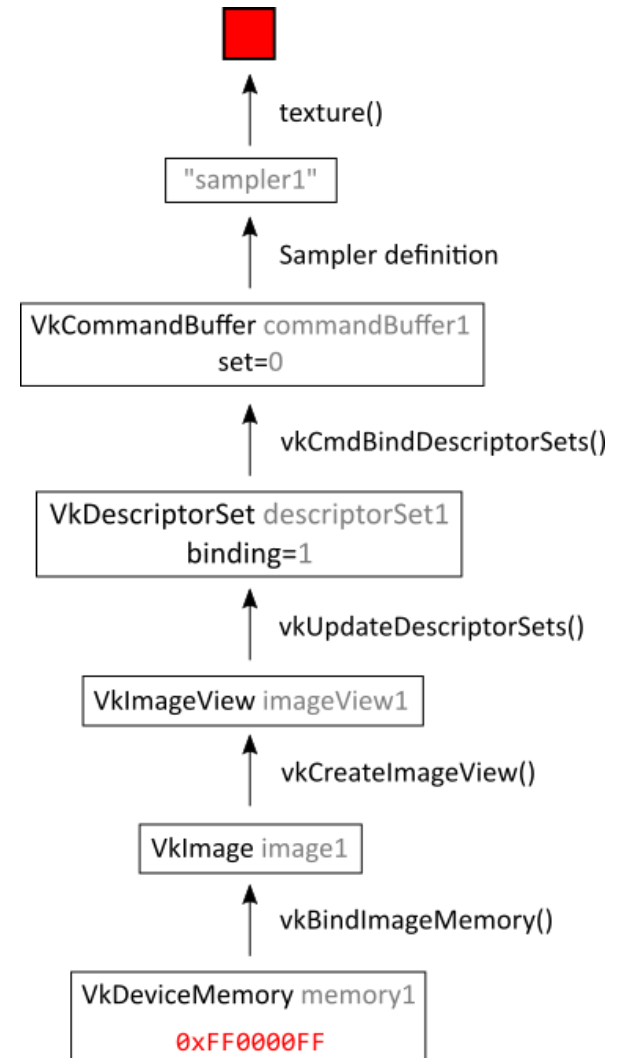


Potok

- Zaleta: brak non-orthogonal states (NOS)
 - Nie ma rekompilacji shadera w niespodziewanych momentach
 - Sterownik nie tworzy własnych wątków, nie robi niczego w tle
 - Przewidywalna wydajność, gra się nie zacina („no hitching”)
- Wada: trzeba utworzyć i zarządzać wszystkimi potrzebnymi potokami
 - Ich tworzenie może być czasochłonne
 - Zmiana jakiegokolwiek parametru wymaga nowego potoku

Deskryptory

- Dostęp z shaderów do zasobów odbywa się za pośrednictwem deskryptorów – **VkDescriptorSet**
- Od shadera do pamięci prowadzi długa droga...
Nie będziemy omawiać całej



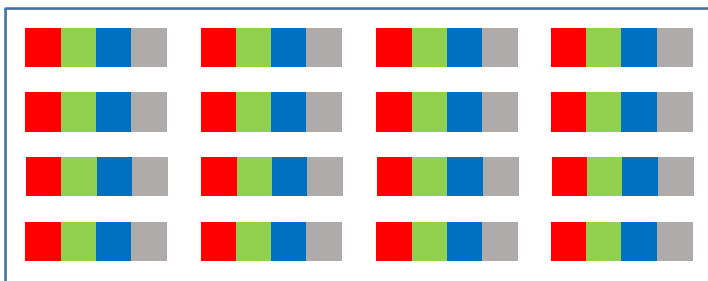
Zasoby

Tylko 2 rodzaje zasobów:

- **VkBuffer** – bufor danych binarnych określonej długości
 - Może służyć jako vertex buffer, index buffer, uniform (constant) buffer i inne...

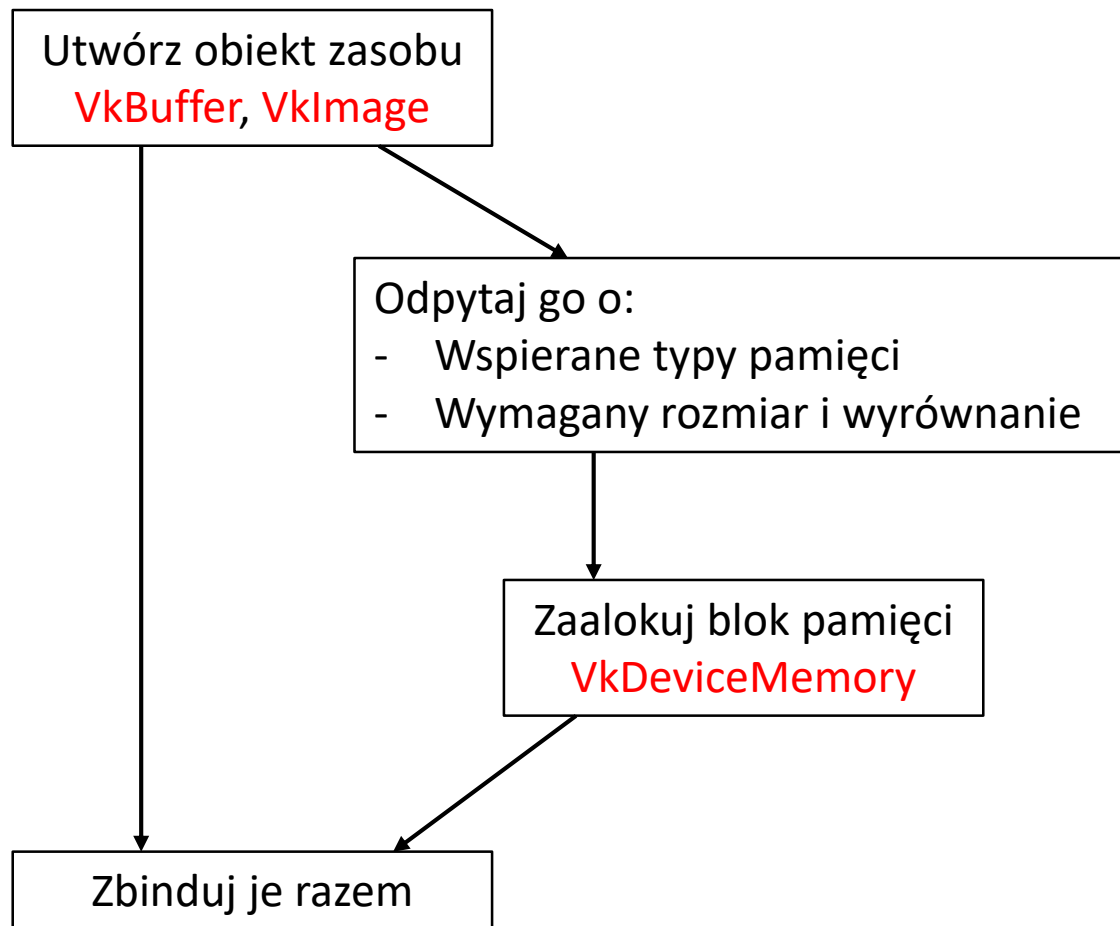


- **VkImage** – tekstura
 - Wiele parametrów: width, height, depth, pixel format, mip levels, array layers, MSAA, ...



Zasoby

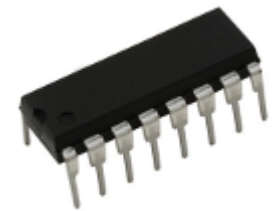
Utworzenie zasobu wymaga kilku etapów:



Pamięć

Sprawdź listę rodzajów pamięci dostępnych w urządzeniu

Inna na kartach dyskretnych/zintegrowanych, inna na AMD/NVIDIA/Intel...



- **VkPhysicalDevice**

- **VkMemoryHeap** ————— **size** – rozmiar w bajtach

- **VkMemoryType** ————— **propertyFlags** – właściwości

- **VkMemoryType**

- **VkMemoryHeap**

- **VkMemoryType**

- **VkMemoryType**

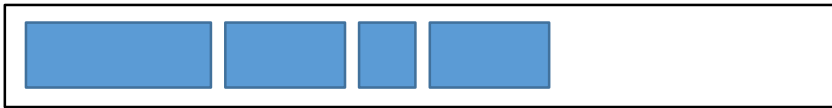
Pamięć

VkMemoryType::propertyFlags

- **VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT**
 - Pamięć lokalna dla GPU – szybki dostęp
 - „Device” znaczy układ graficzny – GPU
- **VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT**
 - Pamięć widoczna dla CPU – można mapować
 - „Host” znaczy procesor główny – CPU
- **VK_MEMORY_PROPERTY_HOST_COHERENT_BIT...**
- **VK_MEMORY_PROPERTY_HOST_CACHED_BIT...**

Alokacja pamięci

- Nie alokuj oddzielnego bloku dla każdego zasobu
- Alokuj większe bloki i przydzielaj zasobom ich fragmenty



- ...lub użyj gotowej biblioteki: Vulkan Memory Allocator
<https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator/>


Autoreklama 😊

Synchronizacja

Rodzaje obiektów synchronizujących:

1. **VkFence** – z GPU do CPU

- Możemy poczekać, aż `VkCommandBuffer` się wykona.




```
VkFence fence;  
vkQueueSubmit(queue, 1, &submitInfo, fence);  
vkWaitForFences(device, 1, &fence, VK_TRUE, UINT64_MAX);
```

Synchronizacja

Rodzaje obiektów synchronizujących:

1. **VkFence** – z GPU do CPU
2. **VkSemaphore** – z jednej do innej kolejki GPU
 - Submit przyjmuje listę WaitSemaphores i SignalSemaphores

```
VkSemaphore semaphore;  
VkSubmitInfo submit1, submit2;  
  
submit1.signalSemaphoreCount = 1;  
submit1.pSignalSemaphores = &semaphore;  
vkQueueSubmit(queue1, 1, &submit1, fence1);  
  
submit2.waitSemaphoreCount = 1;  
submit2.pWaitSemaphores = &semaphore;  
vkQueueSubmit(queue2, 1, &submit2, fence2);
```



Synchronizacja

Rodzaje obiektów synchronizujących:

1. **VkFence** – z GPU do CPU
2. **VkSemaphore** – z jednej do innej kolejki GPU
3. **VkEvent** – w ramach jednej kolejki GPU
4. Bariery

Bariery

- Bariera to polecenie dotyczące określonego zasobu
- Niezbędne do synchronizacji między użyciami tego zasobu

Typowy przykład:

1. Zapis (renderowanie) do **image1** jako „render target” (color attachment)

2. **BARIERA!**



3. Odczyt (sAMPLowanie) z **image1** jako „tekstura” (sampled image)

Bariery

Polecenie wstawiane do VkCommandBuffer między wywołania renderujące.

```
VkImageMemoryBarrier barrier = { VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER };
barrier.srcAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT;
barrier.dstAccessMask = VK_ACCESS_SHADER_READ_BIT;
barrier.oldLayout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;
barrier.newLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
barrier.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
barrier.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
barrier.image = image1;
barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
barrier.subresourceRange.baseMipLevel = 0;
barrier.subresourceRange.levelCount = 1;
barrier.subresourceRange.baseArrayLayer = 0;
barrier.subresourceRange.layerCount = 1;
```

```
vkCmdPipelineBarrier(
    cmdBuf,
    VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT, // srcStageMask
    VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT, // dstStageMask
    0, // dependencyFlags
    0, nullptr, // memoryBarrierCount, pMemoryBarriers
    0, nullptr, // bufferMemoryBarrierCount, pBufferMemoryBarriers
    1, &barrier); // imageMemoryBarrierCount, pImageMemoryBarriers
```

Bariery

Dotyczy konkretnego VkImage i konkretnego zakresu jego subresources.

```
VkImageMemoryBarrier barrier = { VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER };
barrier.srcAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT;
barrier.dstAccessMask = VK_ACCESS_SHADER_READ_BIT;
barrier.oldLayout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;
barrier.newLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
barrier.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
barrier.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
barrier.image = image1;
barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
barrier.subresourceRange.baseMipLevel = 0;
barrier.subresourceRange.levelCount = 1;
barrier.subresourceRange.baseArrayLayer = 0;
barrier.subresourceRange.layerCount = 1;
```

```
vkCmdPipelineBarrier(
    cmdBuf,
    VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT, // srcStageMask
    VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT, // dstStageMask
    0, // dependencyFlags
    0, nullptr, // memoryBarrierCount, pMemoryBarriers
    0, nullptr, // bufferMemoryBarrierCount, pBufferMemoryBarriers
    1, &barrier); // imageMemoryBarrierCount, pImageMemoryBarriers
```

Bariery

3 pary flag.

Tylko niektóre kombinacje mają sens.

```
VkImageMemoryBarrier barrier = { VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER };
barrier.srcAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT;
barrier.dstAccessMask = VK_ACCESS_SHADER_READ_BIT;
barrier.oldLayout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;
barrier.newLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
barrier.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
barrier.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
barrier.image = image1;
barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
barrier.subresourceRange.baseMipLevel = 0;
barrier.subresourceRange.levelCount = 1;
barrier.subresourceRange.baseArrayLayer = 0;
barrier.subresourceRange.layerCount = 1;

vkCmdPipelineBarrier(
    cmdBuf,
    VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT, // srcStageMask
    VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT, // dstStageMask
    0, // dependencyFlags
    0, nullptr, // memoryBarrierCount, pMemoryBarriers
    0, nullptr, // bufferMemoryBarrierCount, pBufferMemoryBarriers
    1, &barrier); // imageMemoryBarrierCount, pImageMemoryBarriers
```

Barier

Użycie przed:

- srcAccessMask = COLOR_ATTACHMENT_WRITE
- oldLayout = COLOR_ATTACHMENT_OPTIMAL
- srcStageMask = COLOR_ATTACHMENT_OUTPUT

```
VkImageMemoryBarrier barrier = { VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER };
barrier.srcAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT;
barrier.dstAccessMask = VK_ACCESS_SHADER_READ_BIT;
barrier.oldLayout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;
barrier.newLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
barrier.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
barrier.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
barrier.image = image1;
barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
barrier.subresourceRange.baseMipLevel = 0;
barrier.subresourceRange.levelCount = 1;
barrier.subresourceRange.baseArrayLayer = 0;
barrier.subresourceRange.layerCount = 1;

vkCmdPipelineBarrier(
    cmdBuf,
    VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT, // srcStageMask
    VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT, // dstStageMask
    0, // dependencyFlags
    0, nullptr, // memoryBarrierCount, pMemoryBarriers
    0, nullptr, // bufferMemoryBarrierCount, pBufferMemoryBarriers
    1, &barrier); // imageMemoryBarrierCount, pImageMemoryBarriers
```

Barier

Użycie po:

- dstAccessMask = SHADER_READ
- newLayout = SHADER_READ_ONLY_OPTIMAL
- dstStageMask = FRAGMENT_SHADER

```
VkImageMemoryBarrier barrier = { VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER };
barrier.srcAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT;
barrier.dstAccessMask = VK_ACCESS_SHADER_READ_BIT;
barrier.oldLayout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;
barrier.newLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
barrier.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
barrier.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
barrier.image = image1;
barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
barrier.subresourceRange.baseMipLevel = 0;
barrier.subresourceRange.levelCount = 1;
barrier.subresourceRange.baseArrayLayer = 0;
barrier.subresourceRange.layerCount = 1;

vkCmdPipelineBarrier(
    cmdBuf,
    VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT, // srcStageMask
    VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT, // dstStageMask
    0, // dependencyFlags
    0, nullptr, // memoryBarrierCount, pMemoryBarriers
    0, nullptr, // bufferMemoryBarrierCount, pBufferMemoryBarriers
    1, &barrier); // imageMemoryBarrierCount, pImageMemoryBarriers
```

Bariery

- Koncepcja bardzo niskopoziomowa
 - W starych API nieobecne, wykonywane automatycznie
- Jednak wciąż wysokopoziomowa – pełni wiele funkcji:
 - Synchronizacja – zaczekanie na zakończenie poprzedniej operacji
 - Zarządzanie pamięcią podręczną – cache(s) flush/invalidate
 - Konwersja formatu kompresji obrazka – layout
- Biblioteka, która to upraszcza:
https://github.com/Tobski/simple_vulkan_synchronization/

Narzędzia

RenderDoc

<https://renderdoc.org/>

- Łapie ramkę
- Podgląd wywołań API, obiektów, stanów, tekstur, siatek...
- Obsługuje Direct3D 11/12, Vulkan, OpenGL, OpenGL ES
- Dobry do debugowania

Narzędzia

The screenshot displays the RenderDoc v1.0 interface for a game scene. The main window shows a 3D render of a dragon with blue and purple energy effects. The interface is divided into several panels:

- Timeline - Frame #8479:** Shows a sequence of events with EID values (7000, 7200, 7400, 76, 7698, 8000, 8200, 8400, 8600, 8800, 9000, 9200, 9400, 9600, 9800, 10000, 10200, 10400, 10600, 10800). A green bar highlights the current frame's usage for `scratchrendertarget_1118301577_1600x900_0_1.vtex`, indicating Reads, Writes, Read/Write, Barriers, and Clears.
- Event Browser:** A table listing events with EID, Name, and Duration (μs). The selected event is `vkCmdDrawIndexed(17847, 1)` at EID 7698.
- API Inspector:** A detailed view of the selected event, showing command buffer details, pipeline layout (4058878), descriptor sets (5740440 and 2726082), and draw index details (17847).
- Texture Viewer:** Shows the current texture being viewed, with channels (RGBA), subresource (Mip 0 - 1600x900), and slice/face information. It includes zoom and overlay controls.
- Inputs/Outputs:** Displays texture inputs and outputs, such as `FS 5 res3927 = tb_colormap` and `FS 7 res5842 = swirl_effect`.
- Pixel Context:** A color-coded grid representing the pixel context of the selected texture.

At the bottom, the status bar indicates the replay context and confirms that the game was loaded without problems.

Źródło: renderdoc.org

Narzędzia

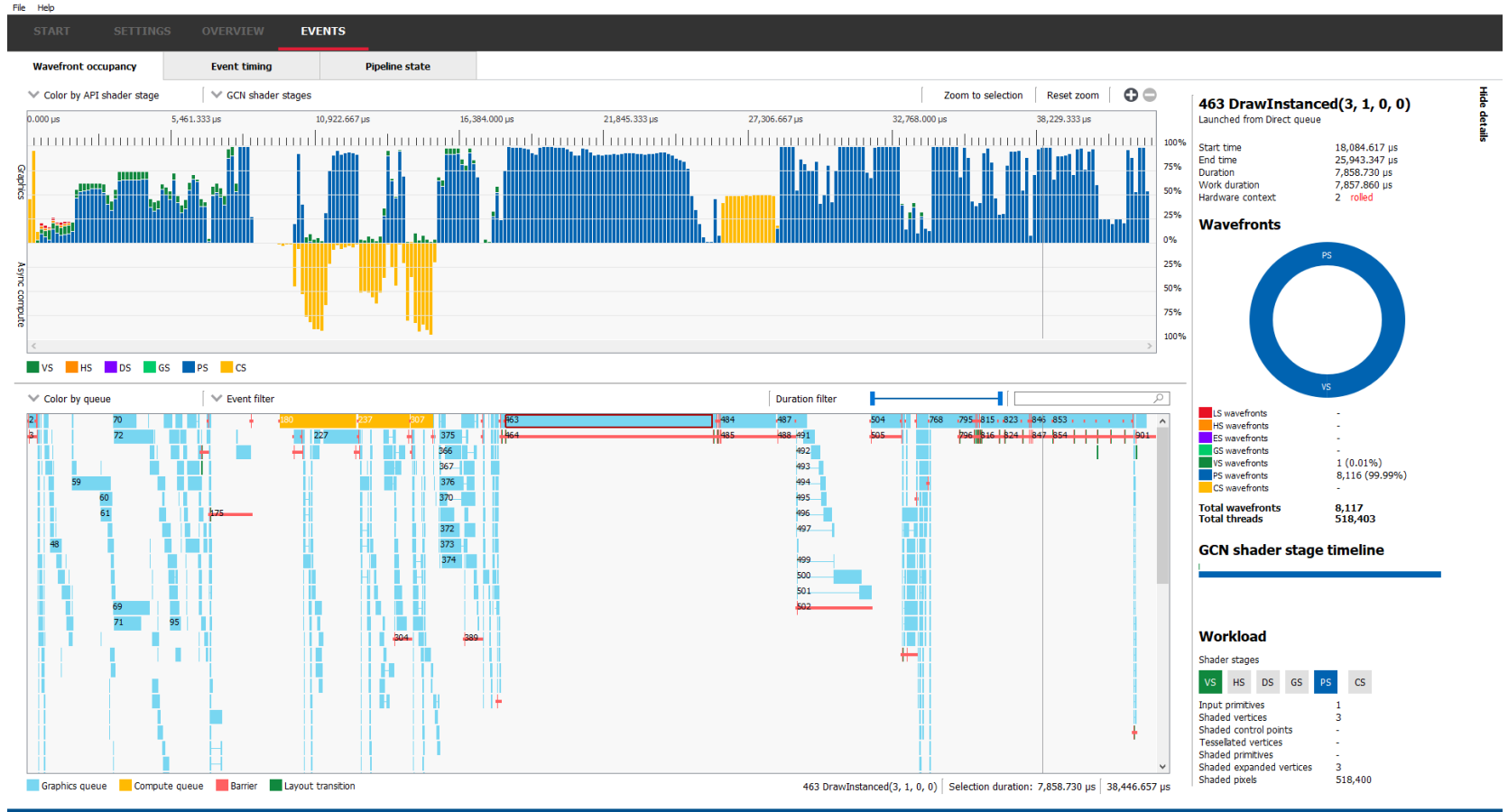
Radeon GPU Profiler (RGP)

<https://github.com/GPUOpen-Tools/Radeon-GPUProfiler>

- Łapie ramkę
- Podgląd wykonania obliczeń na karcie graficznej
- Obsługuje Direct3D 12, Vulkan; wymaga karty AMD
- Dobry do optymalizacji wydajności

Narzędzia

DX12-ExampleTrace.rgp - D3D12 - Radeon GPU Profiler

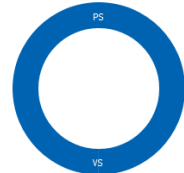


463 DrawInstanced(3, 1, 0, 0)

Launched from Direct queue

Start time 18,084.617 μs
 End time 25,943.347 μs
 Duration 7,858.730 μs
 Work duration 7,857.860 μs
 Hardware context 2 rolled

Wavefronts



Shader stage	Count	Percentage
VS wavefronts	1	0.01%
PS wavefronts	8,116	99.99%
Other wavefronts (HS, DS, GS, CS)	-	-
Total wavefronts	8,117	
Total threads	518,403	

GCN shader stage timeline

Workload

Shader stages: VS HS DS GS PS CS

Input primitives	1
Shaded vertices	3
Shaded control points	-
Tessellated vertices	-
Shaded primitives	-
Shaded expanded vertices	3
Shaded pixels	518,400

Źródło: gpuopen.com

Podsumowanie: Zalety Vulkanu 😊

- Niski poziom, pełna kontrola
 - Można osiągnąć lepszą wydajność
 - Przewidywalna wydajność – nie przycina się
 - Można zoptymalizować pod konkretne (rodzaje) GPU
- Prostszy sterownik
 - Mniejszy narzut na CPU
 - Mniej błędów
- Możliwość zrównoleglenia
 - GPU: wiele kolejek, async compute & transfer
 - CPU: wiele wątków, równoległe wypełnianie VkCommandBuffer
- Obsługa najnowszych funkcji GPU – dzięki rozszerzeniom
- Przenośny na wiele platform
- Inwestycja na przyszłość – zostań specjalistą w wąskiej dziedzinie!

Podsumowanie: Wady Vulkanu ☹️

- Trudniejszy do zrozumienia i opanowania
- Trudniejszy do używania – potrzeba więcej kodu
- Niewybaczący
 - Błędy to teraz zwykle twoja wina
 - Błędny kod na jednych GPU działa, na innych nie

Czy Vulkan jest dla mnie?

Chcę stworzyć grę.



NIE

Unity, Unreal Engine
DirectX 11, OpenGL ES

Interesuję się renderingiem.



TAK

Vulkan, DirectX 12, Metal

Czy Vulkan jest dla mnie?

Chcę stworzyć grę.



NIE

Unity, Unreal Engine
DirectX 11, OpenGL ES

*Chcę się nauczyć
podstaw grafiki.*

Interesuję się renderingiem.



TAK

Vulkan, DirectX 12, Metal

Czy Vulkan jest dla mnie?

Chcę stworzyć grę.

Interesuję się renderingiem.

*Chcę się nauczyć
podstaw grafiki.*

Czy jesteś odważny?

NIE

Unity, Unreal Engine
DirectX 11, OpenGL ES

TAK

Vulkan, DirectX 12, Metal

Literatura

- Vulkan SDK

<https://www.lunarg.com/vulkan-sdk/>

- Khronos Vulkan Registry – dokumentacja

<https://www.khronos.org/registry/vulkan/>

- Understanding Vulkan objects. Adam Sawicki, GPUOpen

<https://gpuopen.com/understanding-vulkan-objects/>

- Vulkan Memory Allocator. GPUOpen

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

Praca

AMD is hiring!

- ☑ Good programmer ?
- ☑ Passionate about graphics ?
- ☑ Willing to work from home + some travel ?
- ☑ Don't mind working for corpo ?

Contact me! <http://asawicki.info>  

Pytania?



Spotkanie
18.04.2018



8170-3849-8100

bit.ly/PolyGami

