An Introduction to Vulkan

Johannes Unterguggenberger
TU Wien, Huawei
PART 1:

Setup
10 min
Starts at 09:00

Lecture
20 min
Starts at 09:10

Coding Session
90 min
Starts at 09:30

PART 2:

Lecture
15 min
Starts at 11:00

Coffee Break
25 min
Starts at 11:15

Coding Session
80 min
Starts at 11:40

Lunch Break 13:00 – 14:00

PART 3:

Lecture
15 min
Starts at 14:00

Coding Session
65 min
Starts at 14:15

Coffee Break
30 min
Starts at 15:20

PART 4:

Lecture
20 min
Starts at 15:50

Coding Session
70 min
Starts at 16:10

Closing
10 min
Starts at 17:20
PART 4

- Command Buffer Allocation
- Memory
- Image Layout Transitions
- Synchronization
PART 4

- Command Buffer Allocation
- Memory
- Image Layout Transitions
- Synchronization
Recap: Command Buffer Recording

QUEUE

COMMAND BUFFER

CMD1  CMD2
Recap: Command Buffer Recording

QUEUE

CMD1

CMD2
Recap: Command Buffer Recording

How to allocate command buffers?
Command Buffer Allocation

```c
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };  
poolCreateInfo.flags = 0; 
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool; 
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };  
allocInfo.commandPool = commandPool; 
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;  
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;  
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };  
beginInfo.flags = 0;  
vkBeginCommandBuffer(commandBuffer, &beginInfo);  
// ...

vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...); 
vkCmdDraw(commandBuffer, ...);  
// ...

vkEndCommandBuffer(commandBuffer);
```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
Command Buffer Allocation

```cpp
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```
Command Buffer Allocation

```c
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
Command Buffer Allocation

```c
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };  
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
   vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };  
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
   vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };  
beginInfo.flags = 0;
   vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
   vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
   vkCmdDraw(commandBuffer, ...);
// ...
   vkEndCommandBuffer(commandBuffer);
```
Command Buffer Allocation

VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO, poolCreateInfo.flags = 0; poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO, allocInfo.commandPool = commandPool; allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO, beginInfo.flags = 0; vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
Command Buffer Allocation

```c
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```
Command Buffer Allocation

```c
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```
Command Buffer Allocation

```c
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };  
.poolCreateInfo.flags = 0;
.poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };  
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };  
.beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```
Queue Submission

vkQueueSubmit

CMD1

CMD2
Queue Submission

Queue Submission

BATCH

CMD1

CMD2

vkQueueSubmit
PART 4

- Command Buffer Allocation
- Memory
- Image Layout Transitions
- Synchronization
PART 4

- Command Buffer Allocation
- Memory
- Image Layout Transitions
- Synchronization
VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT

VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT

VK_MEMORY_PROPERTY_HOST_COHERENT_BIT
(no memory flush needed)
Memory

- VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT
- VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT

Device-Local Memory

Host-Visible Memory

VkBuffer

VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER
Memory

VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER

VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT

VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT

L2

L1

VkBuffer

Device-Local Memory

Host-Visible Memory
Good for data that is updated frequently from the host.
Memory

VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE
VK_DESCRIPTOR_TYPE_STORAGE_IMAGE
VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER

Good for data which should be read fast, or which can be updated on the device

Good for data that is updated frequently from the host

VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT
VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT

VkImage

VkBuffer

Device-Local Memory

Host-Visible Memory

Vulkanised 2023 | An Introduction to Vulkan | TU Wien
PART 4

- Command Buffer Allocation
- Memory
- Image Layout Transitions
- Synchronization
Image Layout Transitions

1) Image into host-visible memory

2) Copy from buffer into image

"Image Layout"
Image Layout Transitions

Note:
Doesn't have to be changing the storage order!
Can also mean some kind of (vendor-specific) compression.
You **don’t** specify the exact memory layout

But the usage scenario

- Concrete memory layout can be vendor-specific
- Concrete transition is implemented in drivers/in hardware
- You specify the **usage scenario**
- Vendor does the right thing
Image Layout Transitions

Image usage descriptions via layouts: \texttt{VkImageLayout}

\begin{verbatim}
    typedef enum VkImageLayout {
        VK_IMAGE_LAYOUT_UNDEFINED = 0,
        VK_IMAGE_LAYOUT_GENERAL = 1,
        VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL = 2,
        VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL = 3,
        VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL = 4,
        VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL = 5,
        VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL = 6,
        VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL = 7,
        ...
    } VkImageLayout;
\end{verbatim}
PART 4

- Command Buffer Allocation
- Memory
- Image Layout Transitions
- Synchronization
PART 4

- Command Buffer Allocation
- Memory
- Image Layout Transitions
- Synchronization
Synchronization

Device-Local Memory

Host-Visible Memory

VKImage

COMMAND BUFFER

CMD 1
CMD 2
CMD 3
CMD 4

memory

write

read

write

write

read and write
Synchronization

Device-Local Memory

Host-Visible Memory

VkImage

COMMAND BUFFER

CMD 1

CMD 2

CMD 3

CMD 4
Synchronization

Device-Local Memory

Host-Visible Memory

VkImage

write

COMMAND BUFFER

CMD 1

CMD 2

CMD 3

CMD 4
Synchronization

Device-Local Memory

Host-Visible Memory

VkImage

COMMAND BUFFER

CMD 1

CMD 2

CMD 3

CMD 4

write

write

write
Synchronization

Device-Local Memory

Host-Visible Memory

VkImage

Command Buffer

CMD 1

CMD 2

CMD 3

CMD 4

read and write

write

write
Synchronization

Device-Local Memory

Host-Visible Memory

COMMAND BUFFER

CMD 1

CMD 2

CMD 3

CMD 4

VkImage

write

write

read

read

write

read and write
Synchronization

Queue: CMD 1, CMD 2, CMD 3, CMD 4

Pipeline Barriers

Memory:
- Device-Local Memory
- Host-Visible Memory

VkImage
Memory Barriers

= Pipeline Barrier + Memory Access
Synchronization

_QUEUE

_BATCH_

_Fence_
Synchronization

QUEUE

BATCH

Fence
Synchronization

**QUEUE**

**BATCH**

CMD 2  CMD 3  CMD 4

**Fence**

CMD X  CMD Y
Synchronization

QUEUE

CMD 2  CMD 3  CMD 4

BATCH

CMD X  CMD Y

Fence
Synchronization

QUEUE

BATCH

CMD 3

CMD 4

CMD X

CMD Y

Fence
Synchronization

QUEUE

BATCH

CMD 3

CMD 4

CMD X

CMD Y

Fence
Synchronization

Fence

vkWaitForFences

waits on

CPU

GPU

signal
Overview of Synchronization Methods

- Wait Idle Operations
- Fences
- Semaphores
  - Binary Semaphores
  - Timeline Semaphores
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events
Image Memory Barriers + Image Layout Transitions

Device-Local Memory

Host-Visible Memory

VkImage

VkBuffer

VK_IMAGE_LAYOUT_UNDEFINED

COMMAND BUFFER

vkCmdCopyBufferToImage

Image Memory Barrier

Image Memory Barrier
Image Memory Barriers + Image Layout Transitions

Device-Local Memory

Host-Visible Memory

VkImage

VkBuffer

COMMAND BUFFER

read and write

Image Memory Barrier

vkCmdCopyBufferToImage

Image Memory Barrier

VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL
Image Memory Barriers + Image Layout Transitions

Device-Local Memory

Host-Visible Memory

VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL

COMMAND BUFFER

Image Memory Barrier

vkCmdCopyBufferToImage

Image Memory Barrier

read and write

read
write
read and write
Image Memory Barriers + Image Layout Transitions

**Device-Local Memory**

**Host-Visible Memory**

**VkImage**

**VkBuffer**

**COMMAND BUFFER**

- `vkCmdCopyBufferToImage`

**Image Memory Barrier**

- `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL`
- `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`

**read and write**

**read**

**write**

**read and write**

---

Vulkanised 2023 | An Introduction to Vulkan | TU Wien
Image Memory Barriers + Image Layout Transitions

Device-Local Memory

Host-Visible Memory

VkImage

VkBuffer

MEMORY

COMMAND BUFFER

read and write

read

write

read and write

vkCmdCopyBufferToImage

Image Memory Barrier

Image Memory Barrier

VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL
PART 4

- Command Buffer Allocation
- Memory
- Image Layout Transitions
- Synchronization
PART 4

- Command Buffer Allocation
- Memory
- Image Layout Transitions
- Synchronization
### Schedule

#### PART 1:
- **Setup**: 10 min
  - Starts at 09:00
- **Lecture**: 20 min
  - Starts at 09:10
- **Coding Session**: 90 min
  - Starts at 09:30

#### PART 2:
- **Lecture**: 15 min
  - Starts at 11:00
- **Coffee Break**: 25 min
  - Starts at 11:15
- **Coding Session**: 80 min
  - Starts at 11:40

**Lunch Break** 13:00 – 14:00

#### PART 3:
- **Lecture**: 15 min
  - Starts at 14:00
- **Coding Session**: 65 min
  - Starts at 14:15
- **Coffee Break**: 30 min
  - Starts at 15:20

#### PART 4:
- **Lecture**: 20 min
  - Starts at 15:50
- **Coding Session**: 70 min
  - Starts at 16:10
- **Closing**: 10 min
  - Starts at 17:20