An Introduction to Vulkan

Lukas Lipp
TU Wien
## Schedule

<table>
<thead>
<tr>
<th>Time</th>
<th>Part</th>
<th>Activity</th>
<th>Duration</th>
<th>Start Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:00</td>
<td>PART 1:</td>
<td>Setup</td>
<td>10 min</td>
<td>09:00</td>
</tr>
<tr>
<td>09:10</td>
<td>PART 1:</td>
<td>Lecture</td>
<td>20 min</td>
<td>09:10</td>
</tr>
<tr>
<td>09:30</td>
<td>PART 1:</td>
<td>Coding Session</td>
<td>90 min</td>
<td>09:30</td>
</tr>
<tr>
<td>11:00</td>
<td>PART 2:</td>
<td>Lecture</td>
<td>15 min</td>
<td>11:00</td>
</tr>
<tr>
<td>11:15</td>
<td>PART 2:</td>
<td>Coffee Break</td>
<td>25 min</td>
<td>11:15</td>
</tr>
<tr>
<td>11:40</td>
<td>PART 2:</td>
<td>Coding Session</td>
<td>80 min</td>
<td>11:40</td>
</tr>
<tr>
<td>17:20</td>
<td>PART 1:</td>
<td>Closing</td>
<td>10 min</td>
<td>17:20</td>
</tr>
<tr>
<td></td>
<td>PART 2:</td>
<td>Lunch Break</td>
<td></td>
<td>13:00 – 14:00</td>
</tr>
<tr>
<td>14:00</td>
<td>PART 3:</td>
<td>Lecture</td>
<td>15 min</td>
<td>14:00</td>
</tr>
<tr>
<td>14:15</td>
<td>PART 3:</td>
<td>Coding Session</td>
<td>65 min</td>
<td>14:15</td>
</tr>
<tr>
<td>15:20</td>
<td>PART 3:</td>
<td>Coffee Break</td>
<td>30 min</td>
<td>15:20</td>
</tr>
<tr>
<td>15:50</td>
<td>PART 4:</td>
<td>Lecture</td>
<td>20 min</td>
<td>15:50</td>
</tr>
<tr>
<td>16:10</td>
<td>PART 4:</td>
<td>Coding Session</td>
<td>70 min</td>
<td>16:10</td>
</tr>
<tr>
<td>17:20</td>
<td>PART 4:</td>
<td>Closing</td>
<td>10 min</td>
<td>17:20</td>
</tr>
</tbody>
</table>
PART 3

- Multiple Vertex Buffers
- Command Buffer Recording
- Multiple Graphics Pipelines
- Depth Test
PART 3

- Multiple Vertex Buffers
- Command Buffer Recording
- Multiple Graphics Pipelines
- Depth Test
Multiple Vertex Buffers

VkBuffer

vkCmdDraw

vertex shader

Draw Processing
Vertex Processing
Tessellation
Primitive Processing
Rasterization

fragment shader

Fragment Processing
Pixel Processing
Multiple Vertex Buffers:
- Positions
- Normals
- Texture Coordinates

vertex shader
- Draw Processing
- Vertex Processing
- Tessellation

fragment shader
- Primitive Processing
- Rasterization
- Fragment Processing
- Pixel Processing
Multiple Vertex Buffers

```cpp
VkVertexInputBindingDescription binding0 = {}; // binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {}; // attribute0.binding = 0;

VkBuffer vertexBuffers[1] = {buffer0};
VkDeviceSize offsets[1] = {0};
vkCmdBindVertexBuffers(commandBuffer, 0, 1, vertexBuffers, offsets);
vkCmdDraw(...);
```
VkVertexInputBindingDescription binding0 = {};  
binding0.binding = 0;  
binding0.stride = sizeof(float) * 3;  
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};  
attribute0.binding = 0;
Multiple Vertex Buffers

```cpp
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputBindingDescription binding1 = {};
binding1.binding = 1;
binding1.stride = sizeof(float) * 3;
binding1.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputBindingDescription binding2 = {};
binding2.binding = 2;
binding2.stride = sizeof(float) * 2;
binding2.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.binding = 0;

VkVertexInputAttributeDescription attribute1 = {};
attribute1.binding = 1;

VkVertexInputAttributeDescription attribute2 = {};
attribute2.binding = 2;

VkBuffer vertexBuffers[3] = {buffer0, buffer1, buffer2};
VkDeviceSize offsets[3] = {0, 0, 0};
vkCmdBindVertexBuffers(commandBuffer, 0, 3, vertexBuffers, offsets);
vkCmdDraw(...);```

Vulkanised 2023 | An Introduction to Vulkan | TU Wien
Multiple Vertex Buffers

```
VkVertexInputBindingDescription binding0 = {};
    binding0.binding = 0;
    binding0.stride = sizeof(float) * 3;
    binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputBindingDescription binding1 = {};
    binding1.binding = 1;
    binding1.stride = sizeof(float) * 3;
    binding1.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputBindingDescription binding2 = {};
    binding2.binding = 2;
    binding2.stride = sizeof(float) * 2;
    binding2.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
    attribute0.binding = 0;

VkVertexInputAttributeDescription attribute1 = {};
    attribute1.binding = 1;

VkVertexInputAttributeDescription attribute2 = {};
    attribute2.binding = 2;

VkBuffer vertexBuffers[3] = {buffer0, buffer1, buffer2};
VkDeviceSize offsets[3] = {0, 0, 0};
vkCmdBindVertexBuffers(commandBuffer, 0, 3, vertexBuffers, offsets);
vkCmdDraw(...);
```
VkVertexInputAttributeDescription attribute0 = {};  
attribute0.location = 0;  
attribute0.binding = 0;  
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;  
attribute0.offset = 0;

VkVertexInputAttributeDescription attribute1 = {};  
attribute1.location = 2;  
attribute1.binding = 1;  
attribute1.format = VK_FORMAT_R32G32B32_SFLOAT;  
attribute1.offset = 0;

VkVertexInputAttributeDescription attribute2 = {};  
attribute2.location = 1;  
attribute2.binding = 2;  
attribute2.format = VK_FORMAT_R32G32_SFLOAT;  
attribute2.offset = 0;

GLSL vertex shader

```glsl
#version 450

layout (binding = 0) uniform UniformBuffer {
    vec4 color;
    mat4 transformationMatrix;
} uniform_buffer;

layout (location = 0) in vec3 in_position;
layout (location = 1) in vec2 in_tex_coord;
layout (location = 2) in vec3 in_normal;

void main() {
    // ...
}
```

Vulkanised 2023 | An Introduction to Vulkan | TU Wien
VkVertexInputAttributeDescription attribute0 = {};  
attribute0.location = 0;  
attribute0.binding = 0;  
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;  
attribute0.offset = 0;

VkVertexInputAttributeDescription attribute1 = {};  
attribute1.location = 2;  
attribute1.binding = 1;  
attribute1.format = VK_FORMAT_R32G32B32_SFLOAT;  
attribute1.offset = 0;

VkVertexInputAttributeDescription attribute2 = {};  
attribute2.location = 1;  
attribute2.binding = 2;  
attribute2.format = VK_FORMAT_R32G32_SFLOAT;  
attribute2.offset = 0;

#version 450
layout (binding = 0) uniform UniformBuffer {
    vec4 color;
    mat4 transformationMatrix;
} uniform_buffer;

layout (location = 0) in vec3 in_position;
layout (location = 1) in vec2 in_tex_coord;
layout (location = 2) in vec3 in_normal;

void main() {
    // ...
}
VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkVertexInputAttributeDescription attribute1 = {};
attribute1.location = 2;
attribute1.binding = 1;
attribute1.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute1.offset = 0;

VkVertexInputAttributeDescription attribute2 = {};
attribute2.location = 1;
attribute2.binding = 2;
attribute2.format = VK_FORMAT_R32G32_SFLOAT;
attribute2.offset = 0;

// GLSL vertex shader

#version 450
layout (binding = 0) uniform UniformBuffer {
    vec4 color;
    mat4 transformationMatrix;
} uniform_buffer;

layout (location = 0) in vec3 in_position;
layout (location = 1) in vec2 in_tex_coord;
layout (location = 2) in vec3 in_normal;

void main() {
    // ...
}
PART 3

- Multiple Vertex Buffers
- Command Buffer Recording
- Multiple Graphics Pipelines
- Depth Test
PART 3

- Multiple Vertex Buffers
- Command Buffer Recording
- Multiple Graphics Pipelines
- Depth Test

The 5th Vulkan Developer Conference
Munich, Germany / February 7–9
Recap: Command Buffer and Descriptors

COMMAND BUFFER

set = 0
Descriptor Set A
Descriptor Set B
vkCmdDraw
vkCmdDraw

set = 1
Descriptor Set C
Recap: Command Buffer and Descriptors

![Diagram showing three descriptor sets and vkCmdDraw commands]
Recap: Command Buffer and Descriptors

**QUEUE**

- `vkCmdDraw`
- `vkCmdDraw`

- **Descriptor Set A**
- **Descriptor Set B**
- **Descriptor Set C**
Recap: Command Buffer and Descriptors

_vkCmdDraw_  
_vkCmdBindDescriptorSets_

_vkCmdDraw_  
_vkCmdBindDescriptorSets_

Descriptor Set A  
Descriptor Set B  
Descriptor Set C

_vkSampler_

_vkBuffer_

_vkImage_

Vulkanised 2023 | An Introduction to Vulkan | TU Wien
VkCommandBuffer command_buffer = /* ... */
VkPipeline pipeline = /* ... */
VkPipelineLayout pipeline_layout = /* ... */
VkDescriptorSet descriptor_set = /* ... */

vkCmdBindDescriptorSets(command_buffer,
    VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline_layout,
    0, 1, &descriptor_set,
    0, nullptr);

vkCmdBindPipeline(command_buffer, VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline);

VkBuffer vertex_buffers[3] {
    buffer0, buffer1, buffer2
};
VkDeviceSize offsets[3]{ 0, 0, 0 };
vkCmdBindVertexBuffer(command_buffer, 0, 3, vertex_buffers, offsets);

vkCmdDraw(command_buffer, 3, 1, 0, 0);
VkCommandBuffer command_buffer = // ...
VkPipeline pipeline = // ...
VkPipelineLayout pipeline_layout = // ...
VkDescriptorSet descriptor_set = // ...

vkCmdBindDescriptorSets(command_buffer,
    VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline_layout,
    0, 1, &descriptor_set,
    0, nullptr);

vkCmdBindPipeline(command_buffer, VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline);

VkBuffer vertex_buffers[3] {
    buffer0, buffer1, buffer2
};
VkDeviceSize offsets[3]{ 0, 0, 0 };
vkCmdBindVertexBuffers(command_buffer, 0, 3, vertex_buffers, offsets);

vkCmdDraw(command_buffer, 3, 1, 0, 0);
VkCommandBuffer command_buffer = // ...
VkPipeline pipeline = // ...
VkPipelineLayout pipeline_layout = // ...
VkDescriptorSet descriptor_set = // ...

vkCmdBindDescriptorSets(command_buffer,
        VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline_layout,
        0, 1, &descriptor_set,
        0, nullptr);

vkCmdBindPipeline(command_buffer, VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline);

VkBuffer vertex_buffers[3] {
        buffer0, buffer1, buffer2
};
VkDeviceSize offsets[3]{ 0, 0, 0 };
vkCmdBindVertexBuffers(command_buffer, 0, 3, vertex_buffers, offsets);

vkCmdDraw(command_buffer, 3, 1, 0, 0);
VkCommandBuffer command_buffer = // ...
VkPipeline pipeline = // ...
VkPipelineLayout pipeline_layout = // ...
VkDescriptorSet descriptor_set = // ...

vkCmdBind(descriptor_set, command_buffer, VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline_layout, 0, 1, &descriptor_set, 0, nullptr);

vkCmdBindPipeline(command_buffer, VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline);

VkBuffer vertex_buffers[3] {
    buffer0, buffer1, buffer2
};
VkDeviceSize offsets[3] { 0, 0, 0 };
vkCmdBindVertexBuffers(command_buffer, 0, 3, vertex_buffers, offsets);

vkCmdDraw(command_buffer, 3, 1, 0, 0);
VkCommandBuffer command_buffer = // ...
VkPipeline pipeline = // ...
VkPipelineLayout pipeline_layout = // ...
VkDescriptorSet descriptor_set = // ...

vkCmdBindDescriptorSets(command_buffer,
    VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline_layout,
    0, 1, &descriptor_set,
    0, nullptr);

vkCmdBindPipeline(command_buffer, VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline);

VkBuffer vertex_buffers[3] {
    buffer0, buffer1, buffer2
};
VkDeviceSize offsets[3]{ 0, 0, 0 };
vkCmdBindVertexBuffers(command_buffer, 0, 3, vertex_buffers, offsets);

vkCmdDraw(command_buffer, 3, 1, 0, 0);
VkCommandBuffer command_buffer = // ...
VkPipeline pipeline = // ...
VkPipelineLayout pipeline_layout = // ...
VkDescriptorSet descriptor_set = // ...

vkCmdBindDescriptorSets(command_buffer,
   VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline_layout,
   0, 1, &descriptor_set,
   0, nullptr);

vkCmdBindPipeline(command_buffer, VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline);

VkBuffer vertex_buffers[3] {
   buffer0, buffer1, buffer2
};
VkDeviceSize offsets[3] { 0, 0, 0 };
vkCmdBindVertexBuffers(command_buffer, 0, 3, vertex_buffers, offsets);

vkCmdDraw(command_buffer, 3, 1, 0, 0);
VkCommandBuffer command_buffer = // ...
VkPipeline pipeline = // ...
VkPipelineLayout pipeline_layout = // ...
VkDescriptorSet descriptor_set = // ...

vkCmdBindDescriptorSets(command_buffer,
    VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline_layout,
    0, 1, &descriptor_set,
    0, nullptr);

vkCmdBindPipeline(command_buffer, VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline);

VkBuffer vertex_buffers[3] {
    buffer0, buffer1, buffer2
};
VkDeviceSize offsets[3]{ 0, 0, 0 };
vkCmdBindVertexBuffer(command_buffer, 0, 3, vertex_buffers, offsets);

vkCmdDraw(command_buffer, 3, 1, 0, 0);
VkCommandBuffer command_buffer = \// ... 
VkPipeline pipeline = \// ...
VkPipelineLayout pipeline_layout = \// ...
VkDescriptorSet descriptor_set = \// ...

vkCmdBindDescriptorSets(command_buffer,
    VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline_layout,
    0, 1, &descriptor_set,
    0, nullptr);

vkCmdBindPipeline(command_buffer, VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline);

VkBuffer vertex_buffers[3] {
    buffer0, buffer1, buffer2
};
VkDeviceSize offsets[3]{ 0, 0, 0 }; 
vkCmdBindVertexBuffer(command_buffer, 0, 3, vertex_buffers, offsets);

vkCmdDraw(command_buffer, 3, 1, 0, 0);
VkCommandBuffer command_buffer = // ...
VkPipeline pipeline = // ...
VkPipelineLayout pipeline_layout = // ...
VkDescriptorSet descriptor_set = // ...

vkCmdBindDescriptorSets(command_buffer,
    VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline_layout,
    0, 1, &descriptor_set,
    0, nullptr);

vkCmdBindPipeline(command_buffer, VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline);

VkBuffer vertex_buffers[3] {
    buffer0, buffer1, buffer2
};
VkDeviceSize offsets[3]{ 0, 0, 0 };
vkCmdBindVertexBuffer(command_buffer, 0, 3, vertex_buffers, offsets);

vkCmdDraw(command_buffer, 3, 1, 0, 0);
VkCommandBuffer command_buffer = // ...
VkPipeline pipeline = // ...
VkPipelineLayout pipeline_layout = // ...
VkDescriptorSet descriptor_set = // ...

vkCmdBindDescriptorSets(command_buffer,
    VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline_layout,
    0, 1, &descriptor_set,
    0, nullptr);

vkCmdBindPipeline(command_buffer, VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline);

VkBuffer vertex_buffers[3] {
    buffer0, buffer1, buffer2
};
VkDeviceSize offsets[3]{ 0, 0, 0 };
VkCmdBindVertexBuffers(command_buffer, 0, 3, vertex_buffers, offsets);

vkCmdDraw(command_buffer, 3, 1, 0, 0);
PART 3

- Multiple Vertex Buffers
- **Command Buffer Recording**
- Multiple Graphics Pipelines
- Depth Test
PART 3

- Multiple Vertex Buffers
- Command Buffer Recording
- Multiple Graphics Pipelines
- Depth Test
One specific graphics pipeline: `VkPipeline pipe1;`

A different, but still specific graphics pipeline: `VkPipeline pipe2;`

Yet another specific graphics pipeline: `VkPipeline pipe3;`
A different, but still specific graphics pipeline: `VkPipeline pipe2;`

- **Draw Processing** (`myShader.vert`)
- **Vertex Processing**
- **Primitive Processing**
- **Rasterization**
- **Fragment Processing** (`myShader.frag`)
- **Pixel Processing**

**Settings:**
- Filled triangles (`VK_POLYGON_MODE_FILL`)
- Culling: Backfaces (`VK_CULL_MODE_BACK_BIT`)

Yet another specific graphics pipeline: `VkPipeline pipe3;`

- **Draw Processing** (`myShader.vert`)
- **Vertex Processing**
- **Primitive Processing**
- **Rasterization**
- **Fragment Processing** (`myShader.frag`)
- **Pixel Processing**

**Settings:**
- Lines/"Wireframe" (`VK_POLYGON_MODE_LINE`)
- Culling: Nothing (`VK_CULL_MODE_NONE`)
A different, but still specific graphics pipeline: \textcolor{red}{\textbf{VkPipeline pipe2;}}

\begin{itemize}
  \item Draw Processing
  \item Vertex Processing
  \item Primitive Processing
  \item Rasterization
  \item Fragment Processing
  \item Settings:
    \begin{itemize}
      \item Filled triangles (VK_POLYGON_MODE_FILL)
      \item Culling: Backfaces (VK_CULL_MODE_BACK_BIT)
    \end{itemize}
\end{itemize}

Yet another specific graphics pipeline: \textcolor{red}{\textbf{VkPipeline pipe3;}}

\begin{itemize}
  \item Draw Processing
  \item Vertex Processing
  \item Primitive Processing
  \item Rasterization
  \item Fragment Processing
  \item Settings:
    \begin{itemize}
      \item Lines/"Wireframe" (VK_POLYGON_MODE_LINE)
      \item Culling: Nothing (VK_CULL_MODE_NONE)
    \end{itemize}
\end{itemize}
PART 3

- Multiple Vertex Buffers
- Command Buffer Recording
- Multiple Graphics Pipelines
- Depth Test
PART 3

- Multiple Vertex Buffers
- Command Buffer Recording
- Multiple Graphics Pipelines
- Depth Test
Without Depth Test

Draw Processing → Vertex Processing → Tessellation → Primitive Processing → Rasterization → Fragment Processing → Pixel Processing
Without Depth Test

Vulkanised 2023 | An Introduction to Vulkan | TU Wien
Without Depth Test
Without Depth Test

No depth buffer used
Solution:
- Use depth buffer image (`VkImage`)
- Image usage: `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- Add to framebuffer (i.e., color attachment and depth attachment)
Solution:
- Use depth buffer image (VkImage)
- Image usage: VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- Add to framebuffer (i.e., color attachment and depth attachment)
Solution:
- Use depth buffer image (VkImage)
- Image usage: VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- Add to framebuffer (i.e., color attachment and depth attachment)
Solution:
- Use depth buffer image (VkImage)
- Image usage: VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- Add to framebuffer (i.e., color attachment and depth attachment)
PART 3

- Multiple Vertex Buffers
- Command Buffer Recording
- Multiple Graphics Pipelines
- Depth Test
<table>
<thead>
<tr>
<th>PART</th>
<th>Activity</th>
<th>Duration</th>
<th>Start Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Setup</td>
<td>10 min</td>
<td>09:00</td>
</tr>
<tr>
<td></td>
<td>Lecture</td>
<td>20 min</td>
<td>09:10</td>
</tr>
<tr>
<td></td>
<td>Coding Session</td>
<td>90 min</td>
<td>09:30</td>
</tr>
<tr>
<td>2</td>
<td>Lecture</td>
<td>15 min</td>
<td>11:00</td>
</tr>
<tr>
<td></td>
<td>Coffee Break</td>
<td>25 min</td>
<td>11:15</td>
</tr>
<tr>
<td></td>
<td>Coding Session</td>
<td>80 min</td>
<td>11:40</td>
</tr>
<tr>
<td></td>
<td>Lunch Break</td>
<td></td>
<td>13:00 – 14:00</td>
</tr>
<tr>
<td>3</td>
<td>Lecture</td>
<td>15 min</td>
<td>14:00</td>
</tr>
<tr>
<td></td>
<td>Coding Session</td>
<td>65 min</td>
<td>14:15</td>
</tr>
<tr>
<td></td>
<td>Coffee Break</td>
<td>30 min</td>
<td>15:20</td>
</tr>
<tr>
<td>4</td>
<td>Lecture</td>
<td>20 min</td>
<td>15:50</td>
</tr>
<tr>
<td></td>
<td>Coding Session</td>
<td>70 min</td>
<td>16:10</td>
</tr>
<tr>
<td></td>
<td>Closing</td>
<td>10 min</td>
<td>17:20</td>
</tr>
</tbody>
</table>