Vulkanised 2025

The 7th Vulkan Developer Conference
Cambridge, UK | February 11-13, 2025

Device-Generated Commands in Vulkan

Ricardo Garcia, Igalia

About me

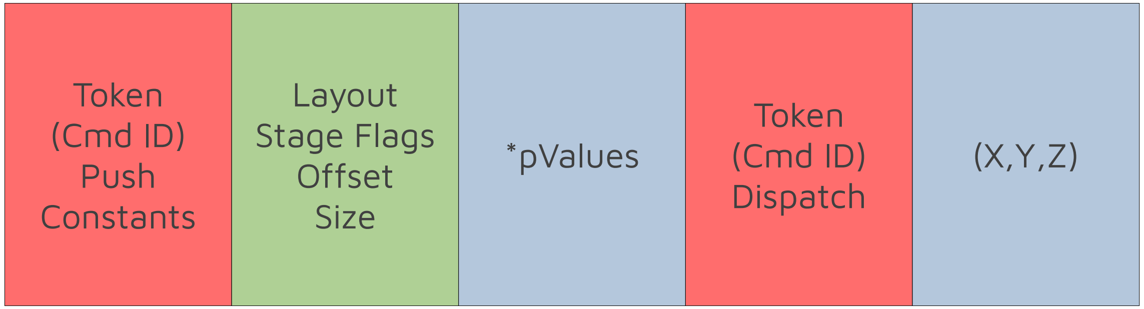
- Part of the Graphics team at Igalia since 2019.
- Focused on Vulkan CTS work for Valve.
- Main author of tests for mesh shading and device-generated commands.

What are Device-Generated Commands?

- One step ahead of indirect draws and dispatches.
- One step behind work graphs.
- Allows drivers to read command sequences from a regular buffer instead of a command buffer.
- That buffer could be filled from the GPU to achieve GPU-driven rendering.
- Better translation of DX12’s ExecuteIndirect.

Naïve CPU-based Approach

1) vkCmdPushConstants(layout, stageFlags, offset, size, pValues)
2) vkCmdDispatch(x, y, z)

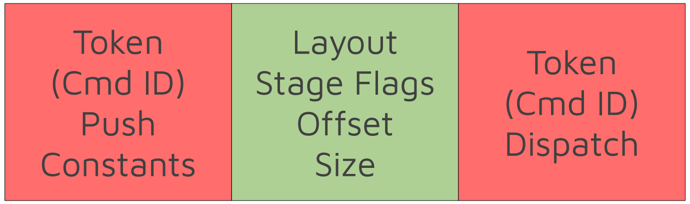


VK_EXT_device_generated_commands

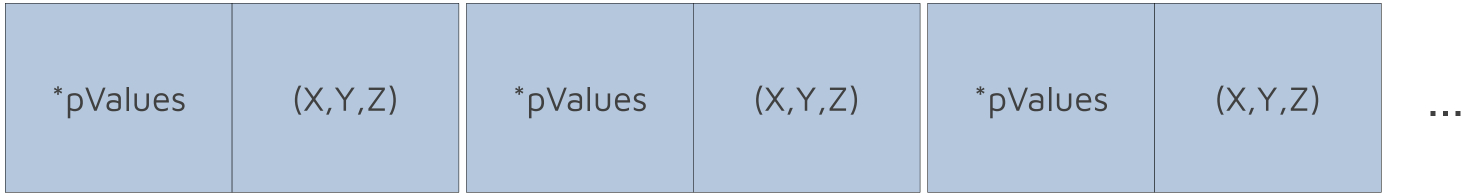
- VkIndirectCommandsLayoutEXT

  1) vkCmdPushConstants

  2) vkCmdDispatch



- Buffer contains a number of fixed-size sequences and each follows the layout



Restriced Command Selection

```
VK_INDIRECT_COMMANDS_TOKEN_TYPE_EXECUTION_SET_EXT
VK_INDIRECT_COMMANDS_TOKEN_TYPE_PUSH_CONSTANT_EXT
VK_INDIRECT_COMMANDS_TOKEN_TYPE_SEQUENCE_INDEX_EXT
```

```
VK_INDIRECT_COMMANDS_TOKEN_TYPE_INDEX_BUFFER_EXT
VK_INDIRECT_COMMANDS_TOKEN_TYPE_VERTEX_BUFFER_EXT
VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_INDEXED_EXT
VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_EXT
VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_INDEXED_COUNT_EXT
VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_COUNT_EXT
```

```
VK_INDIRECT_COMMANDS_TOKEN_TYPE_DISPATCH_EXT
```

```
VK_INDIRECT_COMMANDS_TOKEN_TYPE_TRACE_RAYS2_EXT
```

```
VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_MESH_TASKS_NV_EXT
VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_MESH_TASKS_COUNT_NV_EXT
VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_MESH_TASKS_EXT
VK_INDIRECT_COMMANDS_TOKEN_TYPE_DRAW_MESH_TASKS_COUNT_EXT
```

Indirect Commands Layout

- Backbone of the extension.
- Specifies the layout of each sequence in the buffer.
- Must specify exactly one token to dispatch work at the last position.
- [Optional] Allows you to switch shaders for each sequence.

Indirect Commands Layout

```
struct VkIndirectCommandsLayoutCreateInfoEXT
{
    VkStructureType                          sType;
    const void*                              pNext;
    VkIndirectCommandsLayoutUsageFlagsEXT    flags;
    VkShaderStageFlags                       shaderStages;
    uint32_t                                 indirectStride;
    VkPipelineLayout                         pipelineLayout;
    uint32_t                                 tokenCount;
    const VkIndirectCommandsLayoutTokenEXT*  pTokens;
};
```

Indirect Commands Layout

```
struct VkIndirectCommandsLayoutCreateInfoEXT
{
    VkStructureType                         sType;
    const void*                             pNext;
    VkIndirectCommandsLayoutUsageFlagsEXT   flags;
    VkShaderStageFlags                      shaderStages;
    uint32_t                                indirectStride;
    VkPipelineLayout                        pipelineLayout;
    uint32_t                                tokenCount;
    const VkIndirectCommandsLayoutTokenEXT* pTokens;
};


struct VkIndirectCommandsLayoutTokenEXT
{
    VkStructureType                sType;
    const void*                    pNext;
    VkIndirectCommandsTokenTypeEXT type;
    VkIndirectCommandsTokenDataEXT data;
    uint32_t                       offset;
};
```

Indirect Commands Layout

```
struct VkIndirectCommandsLayoutCreateInfoEXT
{
    VkStructureType                         sType;
    const void*                             pNext;
    VkIndirectCommandsLayoutUsageFlagsEXT   flags;
    VkShaderStageFlags                      shaderStages;
    uint32_t                                indirectStride;
    VkPipelineLayout                        pipelineLayout;
    uint32_t                                tokenCount;
    const VkIndirectCommandsLayoutTokenEXT* pTokens;
};
```

```
struct VkIndirectCommandsLayoutTokenEXT
{
    VkStructureType                 sType;
    const void*                     pNext;
    VkIndirectCommandsTokenTypeEXT  type;
    VkIndirectCommandsTokenDataEXT  data;
    uint32_t                        offset;
};
```

```
union VkIndirectCommandsTokenDataEXT
{
    const VkIndirectCommandsPushConstantTokenEXT* pPushConstant;
    const VkIndirectCommandsVertexBufferTokenEXT* pVertexBuffer;
    const VkIndirectCommandsIndexBufferTokenEXT*  pIndexBuffer;
    const VkIndirectCommandsExecutionSetTokenEXT* pExecutionSet;
};
```

Indirect Execution Sets

- A group of similar pipelines or shader objects.
- All state must be identical (only shaders change).
- Each pipeline/shader has an index in the set.
- The IES is specified beforehand and the DGC buffer contains indices into the set.

Indirect Execution Sets

```
struct VkIndirectExecutionSetCreateInfoEXT
{
    VkStructureType                   sType;
    const void*                       pNext;
    VkIndirectExecutionSetInfoTypeEXT type;
    VkIndirectExecutionSetInfoEXT     info;
};
```

```
union VkIndirectExecutionSetInfoEXT
{
    const VkIndirectExecutionSetPipelineInfoEXT* pPipelineInfo;
    const VkIndirectExecutionSetShaderInfoEXT*   pShaderInfo;
};
```

```
struct VkIndirectExecutionSetPipelineInfoEXT
{
    VkStructureType sType;
    const void*     pNext;
    VkPipeline      initialPipeline;
    uint32_t        maxPipelineCount;
};
```

```
struct VkIndirectExecutionSetShaderInfoEXT
{
    VkStructureType                                   sType;
    const void*                                       pNext;
    uint32_t                                          shaderCount;
    const VkShaderEXT*                                pInitialShaders;
    const VkIndirectExecutionSetShaderLayoutInfoEXT* pSetLayoutInfos;
    uint32_t                                          maxShaderCount;
    uint32_t                                          pushConstantRangeCount;
    const VkPushConstantRange*                        pPushConstantRanges;
};
```

Indirect Execution Sets

- Pipelines and shaders in the set can be updated after creation with **vkUpdateIndirectExecutionSetPipelineEXT** and **vkUpdateIndirectExecutionSetShaderEXT**
- Pipelines and shaders have to be created with a special flag: VK_PIPELINE_CREATE_2_INDIRECT_BINDABLE_BIT_EXT or VK_SHADER_CREATE_INDIRECT_BINDABLE_BIT_EXT.
- The IES token, if present, must appear only once and it must be the first one.

Recap so far

1) The DGC buffer is divided into small chunks called sequences.
2) Each sequence follows a template called Indirect Commands Layout.
3) Each sequence must dispatch work once.
4) You may be able to switch the set of shaders used with each sequence with an Indirect Execution Set (check device properties).

Executing Work with DGC

- Before executing the contents of a DGC buffer, apps need to have bound all the needed state to run those commands.
- That includes the initial pipeline state and shader state (even if they will use an IES!).

Executing Work with DGC

```
void vkCmdExecuteGeneratedCommandsEXT(
    VkCommandBuffer                          commandBuffer,
    VkBool32                                 isPreprocessed,
    const VkGeneratedCommandsInfoEXT*  pGeneratedCommandsInfo);

typedef struct VkGeneratedCommandsInfoEXT {
    VkStructureType                    sType;
    const void*                        pNext;
    VkShaderStageFlags                 shaderStages;
    VkIndirectExecutionSetEXT          indirectExecutionSet;
    VkIndirectCommandsLayoutEXT        indirectCommandsLayout;
    VkDeviceAddress                    indirectAddress;
    VkDeviceSize                       indirectAddressSize;
    VkDeviceAddress                    preprocessAddress;
    VkDeviceSize                       preprocessSize;
    uint32_t                           maxSequenceCount;
    VkDeviceAddress                    sequenceCountAddress;
    uint32_t                           maxDrawCount;
} VkGeneratedCommandsInfoEXT;
```

Executing Work with DGC

```
void vkCmdExecuteGeneratedCommandsEXT(
    VkCommandBuffer                          commandBuffer,
    VkBool32                                 isPreprocessed,
    const VkGeneratedCommandsInfoEXT*        pGeneratedCommandsInfo);

typedef struct VkGeneratedCommandsInfoEXT {
    VkStructureType                 sType;
    const void*                     pNext;
    VkShaderStageFlags              shaderStages;
    VkIndirectExecutionSetEXT       indirectExecutionSet;
    VkIndirectCommandsLayoutEXT     indirectCommandsLayout;
    VkDeviceAddress                 indirectAddress;
    VkDeviceSize                    indirectAddressSize;
    VkDeviceAddress                 preprocessAddress;
    VkDeviceSize                    preprocessSize;
    uint32_t                        maxSequenceCount;
    VkDeviceAddress                 sequenceCountAddress;
    uint32_t                        maxDrawCount;
} VkGeneratedCommandsInfoEXT;
```

Preprocess Buffer

- Some drivers need auxiliary space when processing DGC buffers.
- The amount of space can be queried with **vkGetGeneratedCommandsMemoryRequirementsEXT**.
- Apps need to allocate a buffer with a special flag: VK_BUFFER_USAGE_2_PREPROCESS_BUFFER_BIT_EXT
- Apps need to pass that buffer when executing indirect commands.

Explicit Preprocessing

- Key for performance with some drivers.
- Launched with **vkCmdPreprocessGeneratedCommandsEXT** before executing those same indirect commands.
- Typically submitted in a separate command buffer before the one that contains the execution.
- Layout needs to be created with VK_INDIRECT_COMMANDS_LAYOUT_USAGE_EXPLICIT_PREPROCESS_BIT_EXT.
- Needs the same VkGeneratedCommandsInfoEXT contents, input buffer contents and state between preprocessing and execution.

Explicit Preprocessing (cont.)

```
void vkCmdPreprocessGeneratedCommandsEXT(
    VkCommandBuffer                      commandBuffer,
    const VkGeneratedCommandsInfoEXT* pGeneratedCommandsInfo,
    VkCommandBuffer                      stateCommandBuffer);
```

Explicit Preprocessing (cont.)

```
void vkCmdPreprocessGeneratedCommandsEXT(
    VkCommandBuffer                        commandBuffer,
    const VkGeneratedCommandsInfoEXT*      pGeneratedCommandsInfo,
    VkCommandBuffer                        stateCommandBuffer);
```

Using a command buffer as state for another command... WHAT?!

Explicit Preprocessing (cont.)

```
vkCmdBeginRenderPass(cmdBuffer, …);
vkCmdBindDescriptorSets(cmdBuffer, …);
vkCmdBindPipeline(cmdBuffer, …);
vkCmdSetSomeDynamicState(cmdBuffer, …);
vkCmdPushConstants(cmdBuffer, …);

vkCmdExecuteGeneratedCommands(cmdBuffer,
                              VK_TRUE,
                              &genCmdsInfo);

...
```

```
vkBeginCommandBuffer(preprocessCmdBuffer, …);
vkCmdPreprocessGeneratedCommandsEXT(
    preprocessCmdBuffer,
    &genCmdsInfo,
    cmdBuffer);
<synchronization commands>
vkEndCommandBuffer(preprocessCmdBuffer,…);
```

Synchronization

- From preparing (filling) the DGC buffer to executing the commands stored in it.
  - Source Stage: whichever fills the buffer.
  - Source Access: some kind of write.
  - Destination Stage:
    - VK_PIPELINE_STAGE_COMMAND_PREPROCESS_BIT_EXT or
    - VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT.
  - Destination Access:
    - VK_ACCESS_COMMAND_PREPROCESS_READ_BIT_EXT or
    - VK_ACCESS_INDIRECT_COMMAND_READ_BIT

Synchronization (cont.)

- From preprocessing to execution.
  - Source Stage: VK_PIPELINE_STAGE_COMMAND_PREPROCESS_BIT_EXT
  - Source Access: VK_ACCESS_COMMAND_PREPROCESS_WRITE_BIT_EXT
  - Destination Stage: VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT
  - Destination Access: VK_ACCESS_INDIRECT_COMMAND_READ_BIT

Quick How-To

1) Create the commands layout, and IES if needed (VkIndirectCommandsLayoutEXT, VkIndirectExecutionSetEXT)
2) Establish the maximum number of sequences
3) Query the required preprocess buffer size (vkGetGeneratedCommandsMemoryRequirementsEXT)
4) Allocate DGC buffer and preprocess buffer
5) Record commands and state almost normally (including work that fills the DGC buffer)
6) Dispatch work with vkCmdExecuteGeneratedCommandsEXT
7) If using explicit preprocessing (e.g. Proton does it to improve performance):
    a) Use a separate command buffer for it
    b) Pass the main command buffer in as state
    c) Call vkCmdPreprocessGeneratedCommandsEXT and submit this work first, synchronizing with vkCmdExecuteGeneratedCommandsEXT

Thanks for watching!

Join us!

https://www.igalia.com/jobs