Creating a Pipeline with Dynamically Changeable State Variables

The graphics pipeline is full of state information, and, as previously-discussed, is immutable, that is, the information contained inside it is fixed, and can only be changed by creating a new graphics pipeline with new information.

That isn’t quite true. To a certain extent, you can declare parts of the pipeline state changeable. This allows you to change pipeline information on the fly.

This is useful for managing state information that needs to change frequently. This also creates possible optimization opportunities for the Vulkan driver.

Which Pipeline State Variables can be Changed Dynamically

The possible uses for dynamic variables are shown in the \texttt{VkDynamicState} enum:

- VkDynamicState
- VkDynamicStateViewport
- VkDynamicStateLineWidth
- VkDynamicStateDepthBias
- VkDynamicStateBlendConstants
- VkDynamicStateDepthBounds
- VkDynamicStateStencilCompareMask
- VkDynamicStateStencilWriteMask
- VkDynamicStateStencilReference

Creating a Pipeline

\begin{lstlisting}[language=C]
VkDynamicState vds[] = {
    VK_DYNAMIC_STATE_VIEWPORT,
    VK_DYNAMIC_STATE_LINE_WIDTH
};

 VkPipelineDynamicStateCreateInfo vpdsci,
     vpdsci.sType = VK_STRUCTURE_TYPE_PIPELINE_DYNAMIC_STATE_CREATE_INFO;
     vpdsci.pNext = nullptr;
     vpdsci.flags = 0;
     vpdsci.dynamicStateCount = sizeof(vds) / sizeof(VkDynamicState);
     vpdsci.pDynamicStates = &vds;

 VkGraphicsPipelineCreateInfo vgpci,
     vgpci.pDynamicState = &vpdsci;

 vkCreateGraphicsPipelines( LogicalDevice, pipelineCache, 1, &vgpci, PALLOCATOR, &GraphicsPipeline );
\end{lstlisting}

If you declare certain state variables to be dynamic like this, then you \textit{must} fill them in the command buffer! Otherwise, they are undefined.
Creating a Pipeline

 VkPipelineCreateInfo
 - Shader stages
 - VertexInput State
 - InputAssembly State
 - Tesselation State
 - Viewport State
 - Rasterization State
 - MultiSample State
 - DepthStencil State
 - ColorBlend State
 - Dynamic State
 - Pipeline layout

RenderPass
- basePipelineHandle
- basePipelineIndex

 VkPipelineShaderStageCreateInfo

 VkPipelineVertexInputStateCreateInfo

 VkVertexInputBindingDescription

 VkViewportStateCreateInfo

 - Viewport
  - x, y, w, h
  - minDepth
  - maxDepth
  - offset
  - extent

Scissor

 VkPipelineRasterizationStateCreateInfo

 - cullMode
 - polygonMode
 - frontFace
 - lineWidth

 VkSpecializationInfo

 - which stage (VERTEX, etc.)

 VkShaderModule

 VkPipelineInputAssemblyStateCreateInfo

 - Topology

 VkPipelineMultisampleStateCreateInfo

 VkVertexInputAttributeDescription

 - binding
 - stride
 - inputRate

 VkVertexInputIndirectDescription

 VkPipelineViewportStateCreateInfo

 VkViewport

 VkPipelineColorBlendStateCreateInfo

 - depthTestEnable
 - depthWriteEnable
 - depthCompareOp
 - stencilTestEnable
 - stencilOpStateFront
 - stencilOpStateBack
 - blendEnable

 VkPipelineColorBlendAttachmentState

 - blendEnable
 - srcColorBlendFactor
dstColorBlendFactor
 - colorBlendOp
 - srcAlphaBlendFactor
dstAlphaBlendFactor
 - alphaBlendOp
 - colorWriteMask

 VkPipelineDynamicStateCreateInfo

 - Array naming the states that can be set dynamically

Filling State Variables in the Command Buffer

The command buffer-bound function calls to set these dynamic states are:

- vkCmdSetViewport( commandBuffer, firstViewport, viewportCount, pViewports );
- vkCmdSetScissor( commandBuffer, firstScissor, scissorCount, pScissors );
- vkCmdSetLineWidth( commandBuffer, linewidth );
- vkCmdSetDepthBias( commandBuffer, depthBiasConstantFactor, depthBiasClamp, depthBiasSlopeFactor );
- vkCmdSetBlendConstants( commandBuffer, blendConstants[4] );
- vkCmdSetDepthBounds( commandBuffer, minDepthBounds, maxDepthBounds );
- vkCmdSetStencilCompareMask( commandBuffer, faceMask, compareMask );
- vkCmdSetStencilWriteMask( commandBuffer, faceMask, writeMask );
- vkCmdSetStencilReference( commandBuffer, faceMask, reference );