Dynamic 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:

\begin{verbatim}
VK_DYNAMIC_STATE_VIEWPORT
VK_DYNAMIC_STATE_SCISSOR
VK_DYNAMIC_STATE_LINE_WIDTH
VK_DYNAMIC_STATE_DEPTH_BIAS
VK_DYNAMIC_STATE_BLEND_CONSTANTS
VK_DYNAMIC_STATE_DEPTH_BOUNDS
VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK
VK_DYNAMIC_STATE_STENCIL_WRITE_MASK
VK_DYNAMIC_STATE_STENCIL_REFERENCE
\end{verbatim}
Creating a Pipeline

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 );

If you declare certain state variables to be dynamic like this, then you must fill them in the command buffer! Otherwise, they are undefined and bad things are likely to happen.
Creating a Pipeline

 VkGraphicsPipelineCreateInfo

 - Shader stages
 - VertexInput State
 - InputAssembly State
 - Tessellation State
 - Viewport State
 - Rasterization State
 - MultiSample State
 - DepthStencil State
 - ColorBlend State
 - Dynamic State
 - Pipeline layout
 - RenderPass
 - basePipelineHandle
 - basePipelineIndex

 VkPipelineShaderStageCreateInfo

 - which stage (VERTEX, etc.)

 VkPipelineInputBindingDescription

 - binding
 - stride
 - inputRate

 VkPipelineInputAttributeDescription

 - location
 - binding
 - format
 - offset

 VkPipelineVertexInputStateCreateInfo

 - VkPipelineRasterizationStateCreateInfo

 - cullMode
 - polygonMode
 - frontFace
 - lineWidth

 VkPipelineInputAssemblyStateCreateInfo

 - Topology

 VkPipelineViewportStateCreateInfo

 - Viewport
 - Scissor

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

 VkPipelineDynamicStateCreateInfo

 - Array naming the states that can be set dynamically

 VkPipelineColorBlendStateCreateInfo

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

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

 VkPipelineDepthStencilStateCreateInfo

 - VkPipelineColorBlendAttachmentState

 VkCreateGraphicsPipeline()
Filling State Variables in the Command Buffer

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

```c
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 );
```