What is the Vulkan Graphics Pipeline?

Here's what you need to know:

1. The Vulkan Graphics Pipeline is like what OpenGL would call "The State", or "The Context".
2. There's a lot that goes into it.
3. For the most part, the Graphics Pipeline is meant to be immutable – that is, once this combination of state variables is combined into a Pipeline, that Pipeline never gets changed. To make new combinations of state variables, create a new Graphics Pipelines.
4. The shaders get compiled the rest of the way when their Graphics Pipeline gets created.
The GPU and Driver specify the Pipeline Stages – the Vulkan Graphics Pipeline declares what goes in them.

- **Vertex Shader module**
  - Specialization info
  - Vertex Input binding
  - Vertex Input attributes

- **Topology**

- **Viewport**

- **Tessellation Shaders, Geometry Shaders**

- **Depth Clamping**
  - DiscardEnable
  - PolygonMode
  - CullMode
  - FrontFace
  - LineWidth

- **Which states are dynamic**
  - DepthTestEnable
  - DepthWriteEnable
  - DepthCompareOp
  - StencilTestEnable

- **Viewport**

- **Scissoring**

- **Depth Clamping**

- **DiscardEnable**

- **PolygonMode**

- **CullMode**

- **FrontFace**

- **LineWidth**

- **Rasterization**

- **Dynamic State**

- **Depth/Stencil**
  - DepthWriteEnable

- **Tesselation, Geometry Shaders**

- **Tessellation Shaders, Geometry Shader**

- **Color Blending Stage**

- **Color Blending parameters**

---

**The First Step: Create the Graphics Pipeline Layout**

The Graphics Pipeline Layout is fairly static. Only the layout of the Descriptor Sets and information on the Push Constants need to be supplied.

```c
VkResult
Init14GraphicsPipelineLayout( )
{
    VkResult result;
    VkPipelineLayoutCreateInfo vplci;
    vplci.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
    vplci.pNext = nullptr;
    vplci.flags = 0;
    vplci.setLayoutCount = 4;
    vplci.pSetLayouts = &DescriptorSetLayouts[0];
    vplci.pushConstantRangeCount = 0;
    vplci.pPushConstantRanges = (VkPushConstantRange *)nullptr;

    result = vkCreatePipelineLayout( LogicalDevice, IN &vplci, PALLOCATOR, OUT &GraphicsPipelineLayout );
    return result;
}
```

---

*Oregon State University*  
*Computer Graphics*
Vulkan: A Pipeline Records the Following Items:

- Pipeline Layout: DescriptorSets, PushConstants
- Which Shaders are going to be used
- Per-vertex input attributes: location, binding, format, offset
- Per-vertex input bindings: binding, stride, inputRate
- Assembly: topology
  - **Viewport**: x, y, w, h, minDepth, maxDepth
  - **Scissoring**: x, y, w, h
  - Rasterization: cullMode, polygonMode, frontFace, lineWidth
- Depth: depthTestEnable, depthWriteEnable, depthCompareOp
- Stencil: stencilTestEnable, stencilOpStateFront, stencilOpStateBack
- Blending: blendEnable, srcColorBlendFactor, dstColorBlendFactor, colorBlendOp, srcAlphaBlendFactor, dstAlphaBlendFactor, alphaBlendOp, colorWriteMask
- DynamicState: which states can be set dynamically (bound to the command buffer, outside the Pipeline)

**Bold/Italic** indicates that this state item can also be set with Dynamic Variables

Creating a Graphics Pipeline from a lot of Pieces
Creating a Typical Graphics Pipeline

These settings seem pretty typical to me. Let’s write a simplified Pipeline-creator that accepts Vertex and Fragment shader modules and the topology, and always uses the settings in red above.

Link in the Shaders

Use one vpssci array member per shader module you are using

Use one vvibd array member per vertex input array-of-structures you are using
Link in the Per-Vertex Attributes

VkVertexInputAttributeDescription vviad[4]; // an array containing one of these per vertex attribute in all bindings
vviad[0].location = 0; // location in the layout
vviad[0].binding = 0; // which binding description this is part of
vviad[0].format = VK_FORMAT_VEC3; // x, y, z
vviad[0].offset = offsetof(struct vertex, position); // 0

VkVertexInputAttributeDescription vviad[1].location = 1;
vviad[1].binding = 0;
vviad[1].format = VK_FORMAT_VEC3; // nx, ny, nz
vviad[1].offset = offsetof(struct vertex, normal); // 12

VkVertexInputAttributeDescription vviad[2].location = 2;
vviad[2].binding = 0;
vviad[2].format = VK_FORMAT_VEC3; // r, g, b
vviad[2].offset = offsetof(struct vertex, color); // 24

VkVertexInputAttributeDescription vviad[3].location = 3;
vviad[3].binding = 0;
vviad[3].format = VK_FORMAT_VEC2; // s, t
vviad[3].offset = offsetof(struct vertex, texCoord); // 36

These are defined at the top of the sample code so that you don’t need to use confusing image-looking formats for positions, normals, and tex coords

Use one vviad array member per element in the struct for the array-of-structures element you are using as vertex input

Declare the binding descriptions and attribute descriptions

DECLARE CHOICES
VK_PRIMITIVE_TOPOLOGY_POINT_LIST
VK_PRIMITIVE_TOPOLOGY_LINE_LIST
VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST
VK_PRIMITIVE_TOPOLOGY_LINE_STRIP
VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP
VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN
VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY
VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY
VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY
VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY

DECLARE THE VERTEX TOPOLOGY

Tesselation Shader info

Geometry Shader info
Options for vpiasci.topology

VK_PRIMITIVE_TOPOLOGY_POINT_LIST

VK_PRIMITIVE_TOPOLOGY_LINE_LIST

VK_PRIMITIVE_TOPOLOGY_LINE_STRIP

VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST

VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP

VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN

What is “Primitive Restart Enable”?

```c
vpiasci.primitiveRestartEnable = VK_FALSE;
```

“Restart Enable” is used with:
- Indexed drawing.
- Triangle Fan and “Strip topologies

If vpiasci.primitiveRestartEnable is VK_TRUE, then a special “index” indicates that the
primitive should start over. This is more efficient than explicitly ending the current
primitive and explicitly starting a new primitive of the same type.

```c
typedef enum VkIndexType
{
    VK_INDEX_TYPE_UINT16 = 0, // 0 – 65,535
    VK_INDEX_TYPE_UINT32 = 1, // 0 – 4,294,967,295
} VkIndexType;
```

If your VkIndexType is VK_INDEX_TYPE_UINT16, then the special index is 0xffff.
If your VkIndexType is VK_INDEX_TYPE_UINT32, it is 0xffffffff.
One Really Good use of Restart Enable is in Drawing Terrain Surfaces with Triangle Strips

Triangle Strip #0:
Triangle Strip #1:
Triangle Strip #2:

VkViewport
vv
vv.x = 0;
vv.y = 0;
vv.width = (float)Width;
vv.height = (float)Height;
vv.minDepth = 0.0f;
vv.maxDepth = 1.0f;

VkRect2D
vr
vr.offset.x = 0;
vr.offset.y = 0;
vr.extent.width = Width;
vr.extent.height = Height;

VkPipelineViewportStateCreateInfo vpvsci
vpvsci.sType = VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_STATE_CREATE_INFO;
vpvsci.pNext = nullptr;
vpvsci.flags = 0;
vpvsci.viewportCount = 1;
vpvsci.pViewports = &vv;
vpvsci.scissorCount = 1;
vpvsci.pScissors = &vr;

Declare the viewport information
Declare the scissoring information
Group the viewport and scissor information together
What is the Difference Between Changing the Viewport and Changing the Scissoring?

Viewporting operates on vertices and takes place right before the rasterizer. Changing the vertical part of the viewport causes the entire scene to get scaled (scrunched) into the viewport area.

Original Image

Scissoring operates on fragments and takes place right after the rasterizer. Changing the vertical part of the scissor causes the entire scene to get clipped where it falls outside the scissor area.

Setting the Rasterizer State

```c
VkPipelineRasterizationStateCreateInfo vprsci;
    vprsci.sType = VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_CREATE_INFO;
    vprsci.pNext = nullptr;
    vprsci.flags = 0;
    vprsci.depthClampEnable = VK_FALSE;
    vprsci.rasterizerDiscardEnable = VK_FALSE;
    vprsci.polygonMode = VK_POLYGON_MODE_FILL;
    #ifdef CHOICES
        VK_POLYGON_MODE_FILL
        VK_POLYGON_MODE_LINE
        VK_POLYGON_MODE_POINT
    #endif
    vprsci.cullMode = VK_CULL_MODE_NONE;   // recommend this because of the projMatrix[1][1] *= -1.;
    #ifdef CHOICES
        VK_CULL_MODE_NONE
        VK_CULL_MODE_FRONT_BIT
        VK_CULL_MODE_BACK_BIT
        VK_CULL_MODE_FRONT_AND_BACK_BIT
    #endif
    vprsci.frontFace = VK_FRONT_FACE_COUNTER_CLOCKWISE;
    #ifdef CHOICES
        VK_FRONT_FACE_COUNTER_CLOCKWISE
        VK_FRONT_FACE_CLOCKWISE
    #endif
    vprsci.depthBiasEnable = VK_FALSE;
    vprsci.depthBiasConstantFactor = 0.f;
    vprsci.depthBiasClamp = 0.f;
    vprsci.depthBiasSlopeFactor = 0.f;
    vprsci.lineWidth = 1.f;
```

Declare information about how the rasterization will take place
vprsci.depthClampEnable = VK_FALSE;

Depth Clamp Enable causes the fragments that would normally have been discarded because they are closer to the viewer than the near clipping plane to instead get projected to the near clipping plane and displayed.

A good use for this is **Polygon Capping**:

The front of the polygon is clipped, revealing to the viewer that this is really a shell, not a solid.

The gray area shows what would happen with depthClampEnable (except it would have been red).

---

vprsci.depthBiasEnable = VK_FALSE;
vprsci.depthBiasConstantFactor = 0.f;
vprsci.depthBiasClamp = 0.f;
vprsci.depthBiasSlopeFactor = 0.f;

Depth Bias Enable allows scaling and translation of the Z-depth values as they come through the rasterizer to avoid Z-fighting.
MultiSampling State

```c
VkPipelineMultisampleStateCreateInfo vpmsci;
vpmsci.sType = VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO;
vpmsci.pNext = nullptr;
vpmsci.flags = 0;
vpmsci.rasterizationSamples = VK_SAMPLE_COUNT_1_BIT;
vpmsci.sampleShadingEnable = VK_FALSE;
vpmsci.minSampleShading = 0;
vpmsci.pSampleMask = (VkSampleMask *)nullptr;
vpmsci.alphaToCoverageEnable = VK_FALSE;
vpmsci.alphaToOneEnable = VK_FALSE;
```

Declare information about how the multisampling will take place.

Color Blending State for each Color Attachment

```c
VkPipelineColorBlendAttachmentState vpcbas;
vpcbas.blendEnable = VK_FALSE;
vpcbas.srcColorBlendFactor = VK_BLEND_FACTOR_SRC_COLOR;
vpcbas.dstColorBlendFactor = VK_BLEND_FACTOR_ONE_MINUS_SRC_COLOR;
vpcbas.colorBlendOp = VK_BLEND_OP_ADD;
vpcbas.srcAlphaBlendFactor = VK_BLEND_FACTOR_ONE;
vpcbas.dstAlphaBlendFactor = VK_BLEND_FACTOR_ZERO;
vpcbas.alphaBlendOp = VK_BLEND_OP_ADD;
vpcbas.colorWriteMask = (VK_COLOR_COMPONENT_R_BIT |
                        VK_COLOR_COMPONENT_G_BIT |
                        VK_COLOR_COMPONENT_B_BIT |
                        VK_COLOR_COMPONENT_A_BIT);
```

Create an array with one of these for each color buffer attachment. Each color buffer attachment can use different blending operations.
Color Blending State for each Color Attachment

```cpp
VkPipelineColorBlendStateCreateInfo vpcbsci;
vpcbsci.sType = VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_STATE_CREATE_INFO;
vpcbsci.pNext = nullptr;
vpcbsci.flags = 0;
vpcbsci.logicOpEnable = VK_FALSE;
vpcbsci.logicOp = VK_LOGIC_OP_COPY;
#ifdef CHOICES
VK_LOGIC_OP_CLEAR
VK_LOGIC_OP_AND
VK_LOGIC_OP_AND_REVERSE
VK_LOGIC_OP_COPY
VK_LOGIC_OP_AND_INVERTED
VK_LOGIC_OP_NO_OP
VK_LOGIC_OP_XOR
VK_LOGIC_OP_OR
VK_LOGIC_OP_NOR
VK_LOGIC_OP_EQUIVALENT
VK_LOGIC_OP_INVERT
VK_LOGIC_OP_OR_REVERSE
VK_LOGIC_OP_COPY_INVERTED
VK_LOGIC_OP_OR_INVERTED
VK_LOGIC_OP_NAND
VK_LOGIC_OP_SET
#endif
vpcbsci.attachmentCount = 1;
vpcbsci.pAttachments = &vpcbas;
vpcbsci.blendConstants[0] = 0;
vpcbsci.blendConstants[1] = 0;
vpcbsci.blendConstants[2] = 0;
vpcbsci.blendConstants[3] = 0;
This controls blending between the output of the fragment shader and the input to the color attachments.
```

Which Pipeline Variables can be Set Dynamically

```cpp
VkDynamicState vds[] = { VK_DYNAMIC_STATE_VIEWPORT, VK_DYNAMIC_STATE_SCISSOR };typedef CHOICES
VK_DYNAMIC_STATE_VIEWPORT -- vkCmdSetViewport() 
VK_DYNAMIC_STATE_SCISSOR -- vkCmdSetScissor() 
VK_DYNAMIC_STATE_LINE_WIDTH -- vkCmdSetLineWidth() 
VK_DYNAMIC_STATE_DEPTH_BIAS -- vkCmdSetDepthBias() 
VK_DYNAMIC_STATE_BLEND_CONSTANTS -- vkCmdSetBlendingConstants() 
VK_DYNAMIC_STATE_DEPTH_BOUNDS -- vkCmdSetDepthBounds() 
VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK -- vkCmdSetStencilCompareMask() 
VK_DYNAMIC_STATE_STENCIL_WRITE_MASK -- vkCmdSetStencilWriteMask() 
VK_DYNAMIC_STATE_STENCIL_REFERENCE -- vkCmdSetStencilReference() 
#endif
VkPipelineDynamicStateCreateInfo vpdsici;
vpdsici.sType = VK_STRUCTURE_TYPE_PIPELINE_DYNAMIC_STATE_CREATE_INFO;
vpdsici.pNext = nullptr;
vpdsici.flags = 0;
vpcbsci.dynamicStateCount = 0; // leave turned off for now
vpdsici.pDynamicStates = vds;
```

Oregon State
University
Computer Graphics

mjb -- January 29, 2018
### Stencil Operations for Front and Back Faces

<table>
<thead>
<tr>
<th>VkStencilOpState</th>
<th>vsosf</th>
<th>// front</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>vsosf.depthFailOp</td>
<td>VK_STENCIL_OP_KEEP; // what to do if depth operation fails</td>
</tr>
<tr>
<td></td>
<td>vsosf.failOp</td>
<td>VK_STENCIL_OP_KEEP; // what to do if stencil operation fails</td>
</tr>
<tr>
<td></td>
<td>vsosf.passOp</td>
<td>VK_STENCIL_OP_KEEP; // what to do if stencil operation succeeds</td>
</tr>
<tr>
<td>#ifdef CHOICES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_STENCIL_OP_KEEP</td>
<td></td>
<td>-- keep the stencil value as it is</td>
</tr>
<tr>
<td>VK_STENCIL_OP_ZERO</td>
<td></td>
<td>-- set stencil value to 0</td>
</tr>
<tr>
<td>VK_STENCIL_OP_REPLACE</td>
<td></td>
<td>-- replace stencil value with the reference value</td>
</tr>
<tr>
<td>VK_STENCIL_OP_INCREMENT_AND_CLAMP</td>
<td></td>
<td>-- increment stencil value</td>
</tr>
<tr>
<td>VK_STENCIL_OP_DECREMENT_AND_CLAMP</td>
<td></td>
<td>-- decrement stencil value</td>
</tr>
<tr>
<td>VK_STENCIL_OP_INVERT</td>
<td></td>
<td>-- bit-invert stencil value</td>
</tr>
<tr>
<td>VK_STENCIL_OP_INCREMENT_AND_WRAP</td>
<td></td>
<td>-- increment stencil value</td>
</tr>
<tr>
<td>VK_STENCIL_OP_DECREMENT_AND_WRAP</td>
<td></td>
<td>-- decrement stencil value</td>
</tr>
<tr>
<td>#endif</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vsosf.compareOp</td>
<td>VK_COMPARE_OP_NEVER;</td>
<td></td>
</tr>
<tr>
<td>#ifdef CHOICES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_COMPARE_OP_NEVER</td>
<td></td>
<td>-- never succeeds</td>
</tr>
<tr>
<td>VK_COMPARE_OP_LESS</td>
<td></td>
<td>-- succeeds if stencil value is &lt; the reference value</td>
</tr>
<tr>
<td>VK_COMPARE_OP_EQUAL</td>
<td></td>
<td>-- succeeds if stencil value is == the reference value</td>
</tr>
<tr>
<td>VK_COMPARE_OP_LESS_OR_EQUAL</td>
<td></td>
<td>-- succeeds if stencil value is &lt;= the reference value</td>
</tr>
<tr>
<td>VK_COMPARE_OP_GREATER</td>
<td></td>
<td>-- succeeds if stencil value is &gt; the reference value</td>
</tr>
<tr>
<td>VK_COMPARE_OP_NOT_EQUAL</td>
<td></td>
<td>-- succeeds if stencil value is != the reference value</td>
</tr>
<tr>
<td>VK_COMPARE_OP_GREATER_OR_EQUAL</td>
<td></td>
<td>-- succeeds if stencil value is &gt;= the reference value</td>
</tr>
<tr>
<td>VK_COMPARE_OP_ALWAYS</td>
<td></td>
<td>-- always succeeds</td>
</tr>
<tr>
<td>#endif</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vsosf.compareMask</td>
<td>~0;</td>
<td></td>
</tr>
<tr>
<td>vsosf.writeMask</td>
<td>~0;</td>
<td></td>
</tr>
<tr>
<td>vsosf.reference</td>
<td>0;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VkStencilOpState</th>
<th>vsosb</th>
<th>// back</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>vsosb.depthFailOp</td>
<td>VK_STENCIL_OP_KEEP;</td>
</tr>
<tr>
<td></td>
<td>vsosb.failOp</td>
<td>VK_STENCIL_OP_KEEP;</td>
</tr>
<tr>
<td></td>
<td>vsosb.passOp</td>
<td>VK_STENCIL_OP_KEEP;</td>
</tr>
<tr>
<td></td>
<td>vsosb.compareOp</td>
<td>VK_COMPARE_OP_NEVER;</td>
</tr>
<tr>
<td></td>
<td>vsosb.compareMask</td>
<td>~0;</td>
</tr>
<tr>
<td></td>
<td>vsosb.writeMask</td>
<td>~0;</td>
</tr>
<tr>
<td></td>
<td>vsosb.reference</td>
<td>0;</td>
</tr>
</tbody>
</table>

### Uses for Stencil Operations

**Polygon edges without Z-fighting**

**Magic Lenses**

---

*Oregon State University Computer Graphics*
Operations for Depth Values

VkPipelineDepthStencilStateCreateInfo

```cpp
vpdssci;
vpdssci.sType = VK_STRUCTURE_TYPE_PIPELINE_DEPTH_STENCIL_STATE_CREATE_INFO;
vpdssci.pNext = nullptr;
vpdssci.flags = 0;
vpdssci.depthTestEnable = VK_TRUE;
vpdssci.depthWriteEnable = VK_TRUE;
vpdssci.depthCompareOp = VK_COMPARE_OP_LESS;
VP_COMPARE_OP_NEVER -- never succeeds
VP_COMPARE_OP_LESS -- succeeds if new depth value is < the existing value
VP_COMPARE_OP_EQUAL -- succeeds if new depth value is == the existing value
VP_COMPARE_OP_LESS_OR_EQUAL -- succeeds if new depth value is <= the existing value
VP_COMPARE_OP_GREATER -- succeeds if new depth value is > the existing value
VP_COMPARE_OP_NOT_EQUAL -- succeeds if new depth value is != the existing value
VP_COMPARE_OP_GREATER_OR_EQUAL -- succeeds if new depth value is >= the existing value
VP_COMPARE_OP_ALWAYS -- always succeeds

vpdssci.depthBoundsTestEnable = VK_FALSE;
vpdssci.front = vsosf;
vpdssci.back = vsosb;
vpdssci.minDepthBounds = 0.;
vpdssci.maxDepthBounds = 1.;
vpdssci.stencilTestEnable = VK_FALSE;
```

VkGraphicsPipelineCreateInfo

```cpp
vgpci;
vgpci.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
vgpci.pNext = nullptr;
vgpci.flags = 0;

#ifdef CHOICES
VK_PIPELINE_CREATE_DISABLE_OPTIMIZATION_BIT
VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT
VK_PIPELINE_CREATE_DERIVATIVE_BIT
#endif

gpci.stageCount = 2;                           // number of stages in this pipeline

gpci.pStages = vpssci;
gpci.pVertexInputState = &vpvisci;
gpci.pInputAssemblyState = &vpiasci;
gpci.pTessellationState = (VkPipelineTessellationStateCreateInfo *)nullptr;
gpci.pViewportState = &vpvsci;
gpci.pRasterizationState = &vprsci;
gpci.pMultisampleState = &vpmsci;
gpci.pDynamicState = &vpdsci;

gpci.layout = IN GraphicsPipelineLayout;
gpci.renderPass = IN RenderPass;
gpci.subpass = 0;                              // subpass number

gpci.basePipelineHandle = (VkPipeline) VK_NULL_HANDLE;
gpci.basePipelineIndex = 0;

result = vkCreateGraphicsPipelines( LogicalDevice, VK_NULL_HANDLE, 1, IN &vgpci,
PALLOCATOR, OUT pGraphicsPipeline );
```

Putting it all Together! (finally...)

Group all of the individual state information and create the pipeline

```
result = vkCreateGraphicsPipelines( LogicalDevice, VK_NULL_HANDLE, 1, IN &vgpci,
PALLOCATOR, OUT pGraphicsPipeline );
```
Later on, we will Bind the Graphics Pipeline to the Command Buffer when Drawing

```c
vkCmdBindPipeline(CommandBuffers[nextImageIndex], VK_PIPELINE_BIND_POINT_GRAPHICS, GraphicsPipeline);
```