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 Pipeline.
4. The shaders get compiled the rest of the way when their Graphics Pipeline gets created.

The First Step: Create the Graphics Pipeline Layout

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
- Rasterization: cullMode, polygonMode, frontFace, lineWidth
- DepthWriteEnable: depthWriteEnable, depthCompareOp
- Blend: blendEnable, blendSrcFactor, blendDestFactor, blendOp, srcColorBlendFactor, dstColorBlendFactor, colorBlendOp
- DynamicState: which states can be set dynamically (bound to the command buffer, outside the Pipeline)

Both Direct3D and OpenGL currently support Push Constants. Vulkan pushes Push Constants outside the Pipeline.
Creating a Typical Graphics Pipeline

Link in the Per-Vertex Attributes

Options for vpiasci.topology

What is “Primitive Restart Enable”?
One Really Good use of Restart Enable is in Drawing Terrain Surfaces with Triangle Strips

Triangle Strip #0:

Triangle Strip #1:

Triangle Strip #2:

...
MultiSampling State

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

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

Which Pipeline Variables can be Set Dynamically?

Which Pipeline Variables can be Set Dynamically?

Stencil Operations for Front and Back Faces

Uses for Stencil Operations

Polygon edges without Z-fighting
Operations for Depth Values

```
vkPipelineDepthStencilStateCreateInfo 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;
vpdssci.minDepthBounds = 0.;
vpdssci.maxDepthBounds = 1.;
vpdssci.stencilTestEnable = VK_FALSE;
```

Operations for Depth Values

```
#ifdef CHOICES
VK_PIPELINE_CREATE_DISABLE_OPTIMIZATION_BIT
VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT
VK_PIPELINE_CREATE_DERIVATIVE_BIT
#endif
```

```
vkGraphicsPipelineCreateInfo vgpci;
vgpci.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
vgpci.pNext = nullptr;
vgpci.flags = 0;
vgpci.stageCount = 2;                           // number of stages in this pipeline
vgpci.pStages = vpssci;
```

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

Later on, we will Bind the Graphics Pipeline to the Command Buffer when Drawing

```
vCmdBindPipeline(CommandBuffers[nextImageIndex], VK_PIPELINE_BIND_POINT_GRAPHICS, GraphicsPipeline);
```