The Vulkan Graphics Pipeline:

1. The Vulkan Graphics Pipeline is like what OpenGL would call "The State", or "The Context". It is a data structure.
2. The Vulkan Graphics Pipeline is not the processes that OpenGL would call "the graphics pipeline".
3. For the most part, the Vulkan Graphics Pipeline Data Structure is 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.

Creating a Graphics Pipeline from a lot of Pieces:

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

Why is this necessary? It is because the Descriptor Sets and Push Constants data structures have different sizes depending on how many of each you have. So, the exact structure of the Pipeline Layout depends on you telling Vulkan about the Descriptor Sets and Push Constants that you will be using.
Creating a Typical Graphics Pipeline

The Shaders to Use

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:
...

What is the Difference Between Changing the Viewport and Changing the Scissoring?

Viewport: 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 (scrunch) into the viewport area.

Scissoring: 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.

What is “Depth Clamp Enable”?

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).

What is “Depth Bias Enable”?

vprsci.depthBiasEnable = VK_FALSE;
vprsci.depthBiasConstantFactor = 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.

Depth Clamping
MultiSampling State

- Declare information about how the multisampling will take place

Color Blending State for each Color Attachment *

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

Raster Operations for each Color Attachment

- This controls blending between the output of the fragment shader and the input to the color attachments.

Which Pipeline Variables can be Set Dynamically

- Just used as an example in the Sample Code

The Stencil Buffer

- Here’s how the Stencil Buffer works:
  1. While drawing into the Render Buffer, you can write values into the Stencil Buffer at the same time.
  2. While drawing into the Render Buffer, you can do arithmetic on values in the Stencil Buffer at the same time.
  3. When drawing into the Render Buffer, you can write-protect certain parts of the Render Buffer based on values that are in the Stencil Buffer.
Using the Stencil Buffer to Create a Magic Lens

1. Clear the SB = 0
2. Write protect the color buffer
3. Fill a square, setting SB = 1
4. Write-enable the color buffer
5. Draw the solids wherever SB == 0
6. Draw the wireframes wherever SB == 1

Using the Stencil Buffer to Perform Polygon Capping

1. Clear the SB = 0
2. Draw the polygons, setting SB = ~SB
3. Draw a large gray polygon across the entire scene wherever SB != 0

Outlining Polygons the Naïve Way

1. Draw the polygons
2. Draw the edges

Using the Stencil Buffer to Better Outline Polygons

for( each polygon )
{ Draw the edges, setting SB = 1
  Draw the polygon wherever SB != 1
  Draw the edges, setting SB = 0
}
Using the Stencil Buffer to Perform Hidden Line Removal

Stencil Operations for Front and Back Faces

Operations for Depth Values

Putting it all Together! (finally...)

Sidebar: What is the Organization of the Pipeline Data Structure?

Later on, we will Bind a Specific Graphics Pipeline Data Structure to the Command Buffer when Drawing