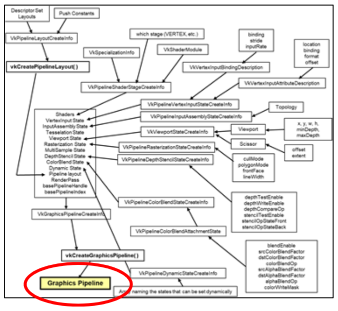
The Graphics Pipeline

Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu

What is the Vulkan Graphics Pipeline?

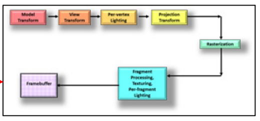


Don't worry if this is too small to read – a larger version is coming up.

There is also a Vulkan Compute Pipeline – we will get to that later.

Here's what you need to know:

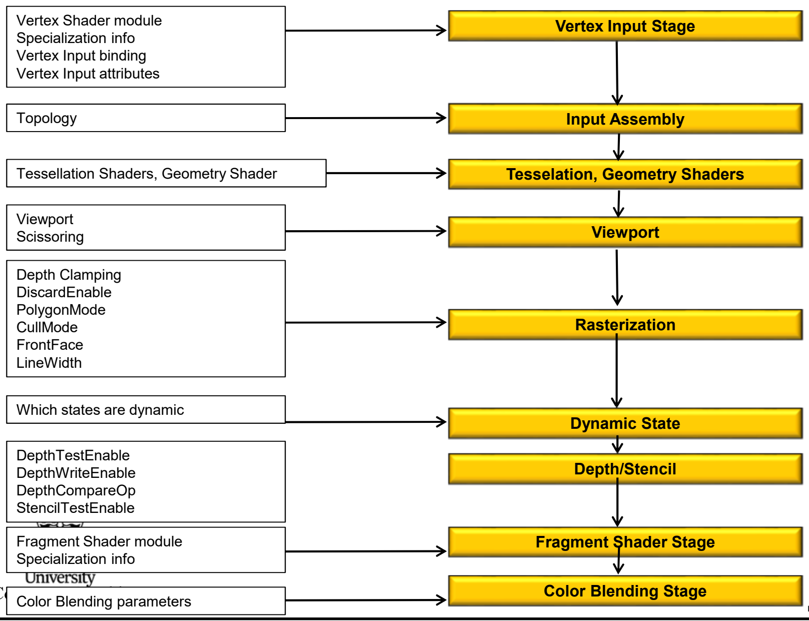
1. The Vulkan Graphics Pipeline is like what OpenGL would call "The State", or "The Context". It is a ***data structure***.
2. The Vulkan Graphics Pipeline is *not* the processes that OpenGL would call "the graphics pipeline".



3. For the most part, the Vulkan Graphics Pipeline 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.

Graphics Pipeline Stages and what goes into Them

The GPU and Driver specify the Pipeline Stages – the
Vulkan Graphics Pipeline declares what goes in them



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.

```
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;
}
```

Let the Pipeline Layout know about the Descriptor Set and Push Constant layouts.

Oregon State
University
Computer Graphics

Vulkan: A Pipeline Records the Following Items:

5

- 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)

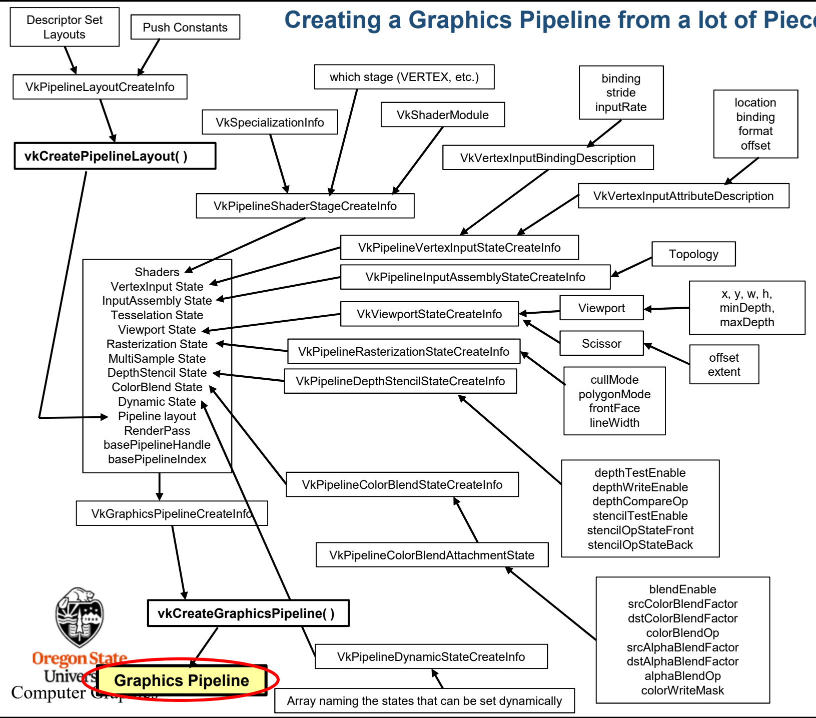
Bold/Italics indicates that this state item can also be set with Dynamic Variables

Oregon State University Computer Graphics

mjb – September 4, 2019

Creating a Graphics Pipeline from a lot of Pieces

6


Descriptor Set Layouts
Push Constants
VkPipelineLayoutCreateInfo
vkCreatePipelineLayout( )
which stage (VERTEX, etc.)
VkSpecializationInfo
VkShaderModule
binding stride inputRate
location binding format offset
VkVertexInputBindingDescription
VkVertexInputAttributeDescription
VkPipelineShaderStageCreateInfo
VkPipelineVertexInputStateCreateInfo
Topology
VkPipelineInputAssemblyStateCreateInfo
x, y, w, h, minDepth, maxDepth
Viewport
VkViewportStateCreateInfo
Scissor
offset extent
VkPipelineRasterizationStateCreateInfo
cullMode polygonMode frontFace lineWidth
VkPipelineDepthStencilStateCreateInfo
Shaders
VertexInput State
InputAssembly State
Tesselation State
Viewport State
Rasterization State
MultiSample State
DepthStencil State
ColorBlend State
Dynamic State
Pipeline layout
RenderPass
basePipelineHandle
basePipelineIndex
depthTestEnable depthWriteEnable depthCompareOp stencilTestEnable stencilOpStateFront stencilOpStateBack
VkPipelineColorBlendStateCreateInfo
VkGraphicsPipelineCreateInfo
VkPipelineColorBlendAttachmentState
blendEnable srcColorBlendFactor dstColorBlendFactor colorBlendOp srcAlphaBlendFactor dstAlphaBlendFactor alphaBlendOp colorWriteMask
vkCreateGraphicsPipeline( )
VkPipelineDynamicStateCreateInfo
Graphics Pipeline
Array naming the states that can be set dynamically

Oregon State University Computer Graphics

mjb – September 4, 2019

Creating a Typical Graphics Pipeline

```
VkResult
Init14GraphicsVertexFragmentPipeline( VkShaderModule vertexShader, VkShaderModule fragmentShader,
                                      VkPrimitiveTopology topology, OUT VkPipeline *pGraphicsPipeline  )
{
#ifdef ASSUMPTIONS
        vvibd[0].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;
        vprsci.depthClampEnable = VK_FALSE;
        vprsci.rasterizerDiscardEnable = VK_FALSE;
        vprsci.polygonMode = VK_POLYGON_MODE_FILL;
        vprsci.cullMode = VK_CULL_MODE_NONE;    // best to do this because of the projectionMatrix[1][1] *= -1.;
        vprsci.frontFace = VK_FRONT_FACE_COUNTER_CLOCKWISE;
        vpmsci.rasterizationSamples = VK_SAMPLE_COUNT_ONE_BIT;
        vpcbas.blendEnable = VK_FALSE;
        vpcbsci.logicOpEnable = VK_FALSE;
        vpdssci.depthTestEnable = VK_TRUE;
        vpdssci.depthWriteEnable = VK_TRUE;
        vpdssci.depthCompareOp = VK_COMPARE_OP_LESS;
#endif
    . . .
```

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.

Oregon State University
Computer Graphics

Link in the Shaders

```
VkPipelineShaderStageCreateInfo                 vpssci[2];
        vpssci[0].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
        vpssci[0].pNext = nullptr;
        vpssci[0].flags = 0;
        vpssci[0].stage = VK_SHADER_STAGE_VERTEX_BIT;
        vpssci[0].module = vertexShader;
        vpssci[0].pName = "main";
        vpssci[0].pSpecializationInfo = (VkSpecializationInfo *)nullptr;
#ifdef BITS
VK_SHADER_STAGE_VERTEX_BIT
VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
VK_SHADER_STAGE_GEOMETRY_BIT
VK_SHADER_STAGE_FRAGMENT_BIT
VK_SHADER_STAGE_COMPUTE_BIT
VK_SHADER_STAGE_ALL_GRAPHICS
VK_SHADER_STAGE_ALL
#endif
        vpssci[1].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
        vpssci[1].pNext = nullptr;
        vpssci[1].flags = 0;
        vpssci[1].stage = VK_SHADER_STAGE_FRAGMENT_BIT;
        vpssci[1].module = fragmentShader;
        vpssci[1].pName = "main";
        vpssci[1].pSpecializationInfo = (VkSpecializationInfo *)nullptr;

    VkVertexInputBindingDescription         vvibd[1];   // an array containing one of these per buffer being used
        vvibd[0].binding = 0;         // which binding # this is
        vvibd[0].stride = sizeof( struct vertex );          // bytes between successive
        vvibd[0].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;
#ifdef CHOICES
VK_VERTEX_INPUT_RATE_VERTEX
VK_VERTEX_INPUT_RATE_INSTANCE
#endif
```

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

Computer Graphics

Link in the Per-Vertex Attributes

9

```
VkVertexInputAttributeDescription          vviad[4];          // an array containing one of these per vertex attribute in all bindings
        // 4 = vertex, normal, color, texture coord
        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
#ifdef EXTRAS_DEFINED_AT_THE_TOP
// these are here for convenience and readability:
#define VK_FORMAT_VEC4          VK_FORMAT_R32G32B32A32_SFLOAT
#define VK_FORMAT_XYZW          VK_FORMAT_R32G32B32A32_SFLOAT
#define VK_FORMAT_VEC3          VK_FORMAT_R32G32B32_SFLOAT
#define VK_FORMAT_STP           VK_FORMAT_R32G32B32_SFLOAT
#define VK_FORMAT_XYZ           VK_FORMAT_R32G32B32_SFLOAT
#define VK_FORMAT_VEC2          VK_FORMAT_R32G32_SFLOAT
#define VK_FORMAT_ST            VK_FORMAT_R32G32_SFLOAT
#define VK_FORMAT_XY            VK_FORMAT_R32G32_SFLOAT
#define VK_FORMAT_FLOAT         VK_FORMAT_R32_SFLOAT
#define VK_FORMAT_S             VK_FORMAT_R32_SFLOAT
#define VK_FORMAT_X             VK_FORMAT_R32_SFLOAT
#endif
        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

        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

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

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

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

University
Computer Graphics

mjb – September 4, 2019

10

```
VkPipelineVertexInputStateCreateInfo          vpvisci;          // used to describe the input vertex attributes
        vpvisci.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
        vpvisci.pNext = nullptr;
        vpvisci.flags = 0;
        vpvisci.vertexBindingDescriptionCount = 1;
        vpvisci.pVertexBindingDescriptions = vvibd;
        vpvisci.vertexAttributeDescriptionCount = 4;
        vpvisci.pVertexAttributeDescriptions = vviad;

    VkPipelineInputAssemblyStateCreateInfo        vpiasci;
        vpiasci.sType = VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO;
        vpiasci.pNext = nullptr;
        vpiasci.flags = 0;
        vpiasci.topology = VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST;;
#ifdef 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_LINE_STRIP_WITH_ADJACENCY
VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY
VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY
#endif
        vpiasci.primitiveRestartEnable = VK_FALSE;

    VkPipelineTessellationStateCreateInfo         vptsci;
        vptsci.sType = VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_STATE_CREATE_INFO;
        vptsci.pNext = nullptr;
        vptsci.flags = 0;
        vptsci.patchControlPoints = 0;      // number of patch control points

    // VkPipelineGeometryStateCreateInfo          vpgsci;
        // vptsci.sType = VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_STATE_CREATE_INFO;
        // vptsci.pNext = nullptr;
        // vptsci.flags = 0;
```

Declare the binding descriptions and attribute descriptions

Declare the vertex topology

Tessellation Shader info

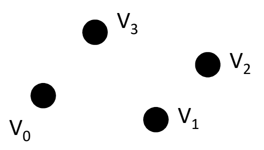
Geometry Shader info

Oreg
Uni
Comput

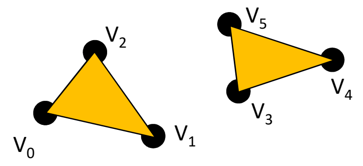
mjb – September 4, 2019

Options for vpiasci.topology

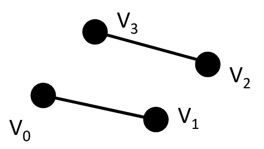
VK_PRIMITIVE_TOPOLOGY_POINT_LIST



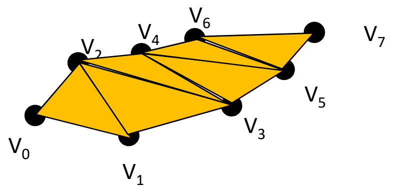
VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST



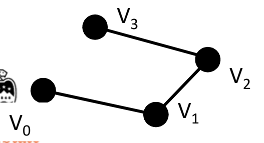
VK_PRIMITIVE_TOPOLOGY_LINE_LIST



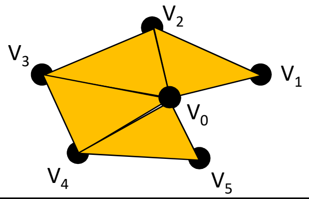
VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP



VK_PRIMITIVE_TOPOLOGY_LINE_STRIP



VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN



Oregon State University Computer Graphics

mjb – September 4, 2019

What is “Primitive Restart Enable”?

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.

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

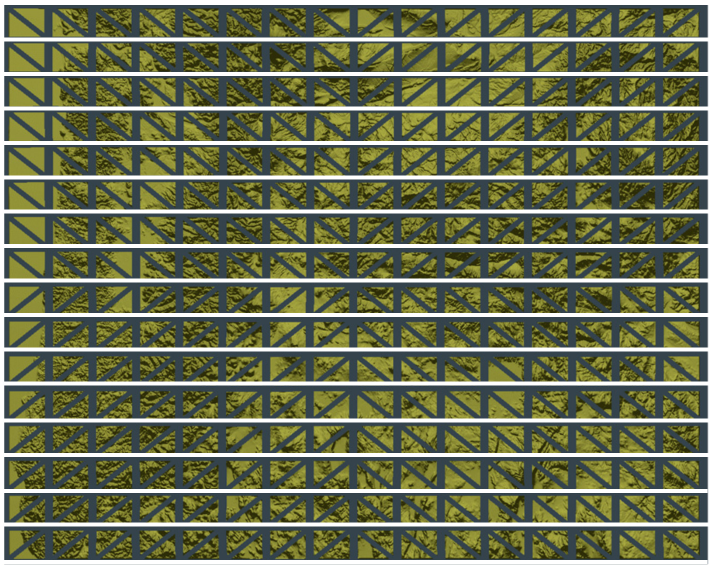
If your VkIndexType is VK_INDEX_TYPE_UINT16, then the special index is 0xffff.
If your VkIndexType is VK_INDEX_TYPE_UINT32, it is 0xffffffff.

Oregon State University Computer Graphics

mjb – September 4, 2019

One Really Good use of Restart Enable is in Drawing Terrain Surfaces with Triangle Strips

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



Oregon State University Computer Graphics

```
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

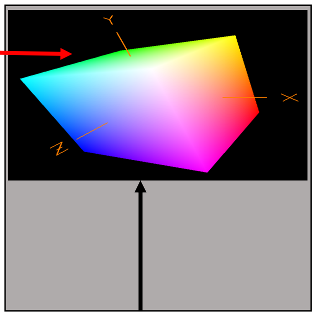
Declare the scissoring information

Group the viewport and scissor information together

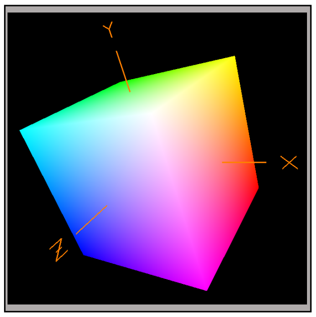
Oregon State University Computer Graphics

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

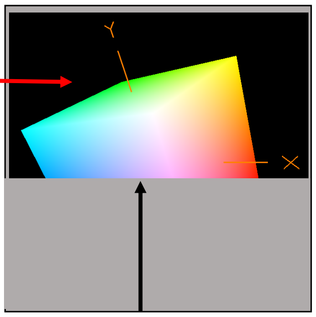
Viewport:
Viewporting operates on ***vertices*** and takes place right before the rasterizer. Changing the vertical part of the **viewport** causes the entire scene to get scaled (scrunched) into the viewport area.



Original Image



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.



Oregon State University
Computer Graphics

Setting the Rasterizer State

```
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

Oregon State University
Computer Graphics

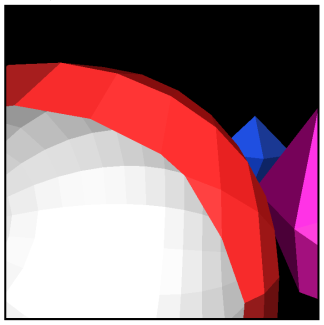
What is "Depth Clamp Enable"?

```
vprsci.depthClampEnable = VK_FALSE;
```

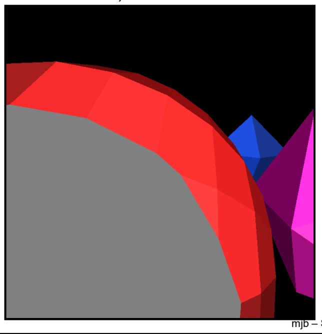
Depth Clamp Enable causes the fragments that would normally have been discarded because they are closer to the viewer than the near clipping plane to instead get projected to the near clipping plane and displayed.

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).

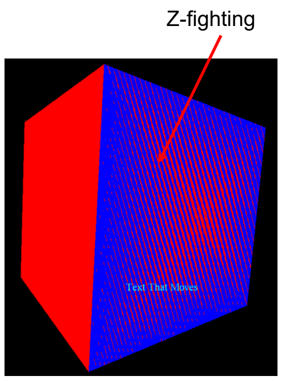


Oregon State University Computer Graphics

What is "Depth Bias Enable"?

```
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.



Oregon State University Computer Graphics

MultiSampling State

```
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

Oregon State University
Computer Graphics

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.

```
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;
```

This controls blending between the output of each color attachment and its image memory.

Oregon State University
Computer Graphics

Color Blending State for each Color Attachment

```
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.

Oregon State University Computer Graphics

Which Pipeline Variables can be Set Dynamically

```
VkDynamicState                  vds[ ] = { VK_DYNAMIC_STATE_VIEWPORT, VK_DYNAMIC_STATE_SCISSOR };
#ifdef 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               -- vkCmdSetBendConstants( )
VK_DYNAMIC_STATE_DEPTH_BOUNDS                  -- vkCmdSetDepthZBounds( )
VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK          -- vkCmdSetStencilCompareMask( )
VK_DYNAMIC_STATE_STENCIL_WRITE_MASK            -- vkCmdSetStencilWriteMask( )
VK_DYNAMIC_STATE_STENCIL_REFERENCE             -- vkCmdSetStencilReferences( )
#endif
      VkPipelineDynamicStateCreateInfo          vpdsci;
            vpdsci.sType = VK_STRUCTURE_TYPE_PIPELINE_DYNAMIC_STATE_CREATE_INFO;
            vpdsci.pNext = nullptr;
            vpdsci.flags = 0;
            vpdsci.dynamicStateCount = 0;             // leave turned off for now
            vpdsci.pDynamicStates = vds;
```

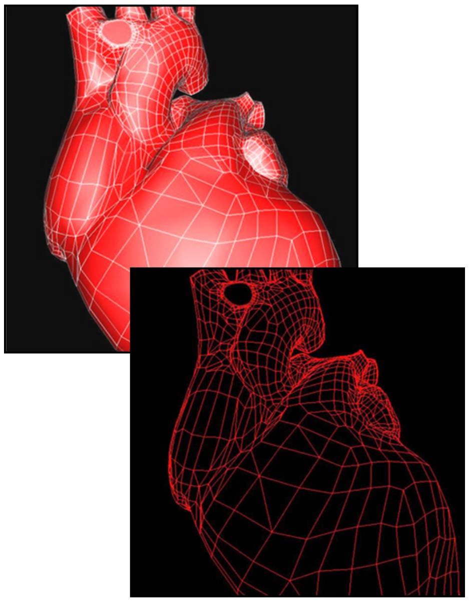
Oregon State University Computer Graphics

Stencil Operations for Front and Back Faces

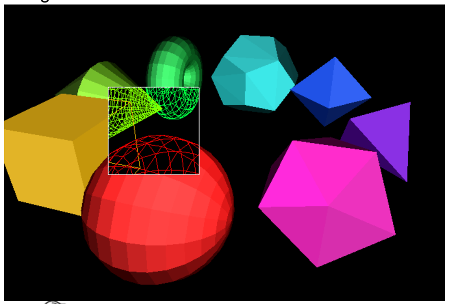
```
        VkStencilOpState                    vsosf;   // front
            vsosf.depthFailOp = VK_STENCIL_OP_KEEP;   // what to do if depth operation fails
            vsosf.failOp        = VK_STENCIL_OP_KEEP;   // what to do if stencil operation fails
            vsosf.passOp        = VK_STENCIL_OP_KEEP;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

        VkStencilOpState                    vsosb;   // back
            vsosb.depthFailOp = VK_STENCIL_OP_KEEP;
            vsosb.failOp        = VK_STENCIL_OP_KEEP;
            vsosb.passOp        = VK_STENCIL_OP_KEEP;
            vsosb.compareOp = VK_COMPARE_OP_NEVER;
            vsosb.compareMask = ~0;
            vsosb.writeMask = ~0;
            vsosb.reference = 0;
```

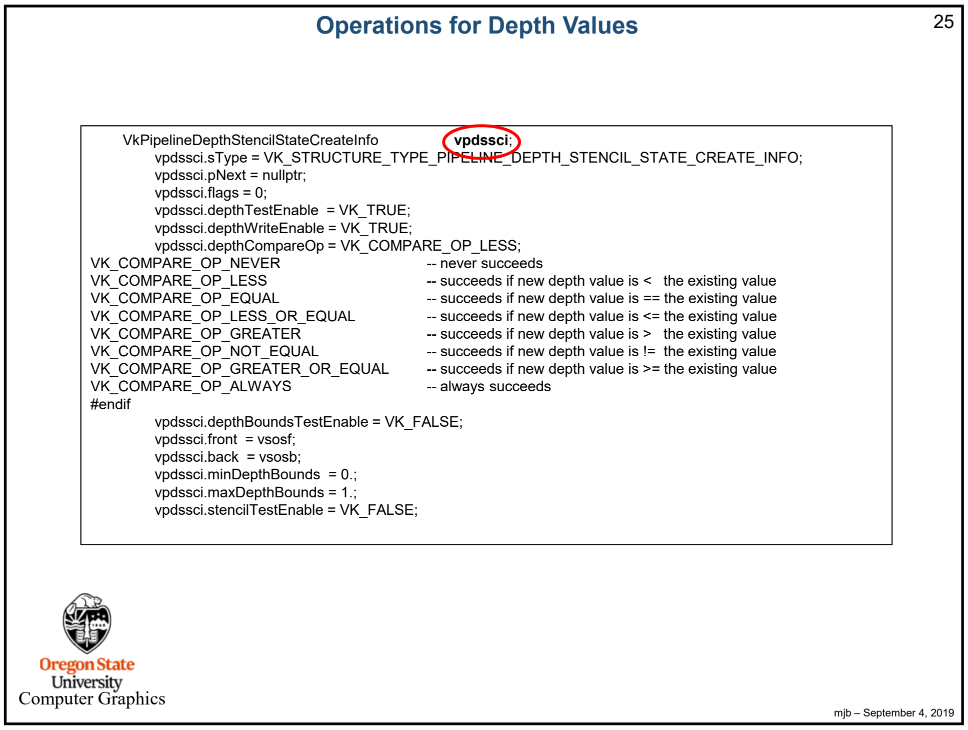
Uses for Stencil Operations

Polygon edges without Z-fighting



Magic Lenses

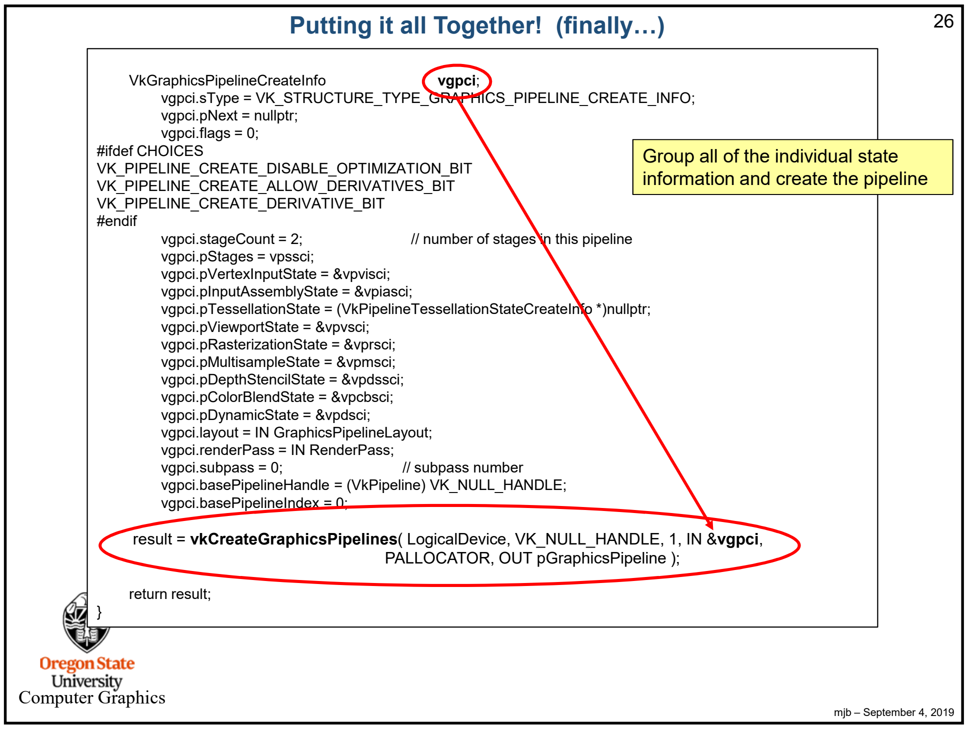




Operations for Depth Values

```
     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;
VK_COMPARE_OP_NEVER                     -- never succeeds
VK_COMPARE_OP_LESS                      -- succeeds if new depth value is <  the existing value
VK_COMPARE_OP_EQUAL                     -- succeeds if new depth value is == the existing value
VK_COMPARE_OP_LESS_OR_EQUAL             -- succeeds if new depth value is <= the existing value
VK_COMPARE_OP_GREATER                   -- succeeds if new depth value is >  the existing value
VK_COMPARE_OP_NOT_EQUAL                 -- succeeds if new depth value is != the existing value
VK_COMPARE_OP_GREATER_OR_EQUAL          -- succeeds if new depth value is >= the existing value
VK_COMPARE_OP_ALWAYS                    -- always succeeds
#endif
          vpdssci.depthBoundsTestEnable = VK_FALSE;
          vpdssci.front  = vsosf;
          vpdssci.back  = vsosb;
          vpdssci.minDepthBounds  = 0.;
          vpdssci.maxDepthBounds = 1.;
          vpdssci.stencilTestEnable = VK_FALSE;
```

Oregon State University
Computer Graphics



Putting it all Together! (finally…)

```
     VkGraphicsPipelineCreateInfo          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
          vgpci.stageCount = 2;                    // number of stages in this pipeline
          vgpci.pStages = vpssci;
          vgpci.pVertexInputState = &vpvisci;
          vgpci.pInputAssemblyState = &vpiasci;
          vgpci.pTessellationState = (VkPipelineTessellationStateCreateInfo *)nullptr;
          vgpci.pViewportState = &vpvsci;
          vgpci.pRasterizationState = &vprsci;
          vgpci.pMultisampleState = &vpmsci;
          vgpci.pDepthStencilState = &vpdssci;
          vgpci.pColorBlendState = &vpcbsci;
          vgpci.pDynamicState = &vpdsci;
          vgpci.layout = IN GraphicsPipelineLayout;
          vgpci.renderPass = IN RenderPass;
          vgpci.subpass = 0;                       // subpass number
          vgpci.basePipelineHandle = (VkPipeline) VK_NULL_HANDLE;
          vgpci.basePipelineIndex = 0;

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

     return result;
}
```

Group all of the individual state information and create the pipeline

Oregon State University
Computer Graphics

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

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