



Descriptor Sets



Oregon State
University

Mike Bailey

mjb@cs.oregonstate.edu



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/)



Oregon State
University
Computer Graphics

In OpenGL

OpenGL puts all uniform data in the same “set”, but with different binding numbers, so you can get at each one.

Each uniform variable gets updated one-at-a-time.

Wouldn't it be nice if we could update a collection of related uniform variables all at once, without having to update the uniform variables that are not related to this collection?

```
layout( std140, binding = 0 ) uniform mat4    uModelMatrix;
layout( std140, binding = 1 ) uniform mat4    uViewMatrix;
layout( std140, binding = 2 ) uniform mat4    uProjectionMatrix;
layout( std140, binding = 3 ) uniform mat3    uNormalMatrix;
layout( std140, binding = 4 ) uniform vec4    uLightPos;
layout( std140, binding = 5 ) uniform float   uTime;
layout( std140, binding = 6 ) uniform int     uMode;
layout(          binding = 7 ) uniform sampler2D uSampler;
```

std140 has to do with the alignment of the different data types. It is the simplest, and so we use it in class to give everyone the highest probability that their system will be compatible with the alignment.



What are Descriptor Sets?

Descriptor Sets are an intermediate data structure that tells shaders how to connect information held in GPU memory to groups of related uniform variables and texture sampler declarations in shaders. There are three advantages in doing things this way:

- Related uniform variables can be updated as a group, gaining efficiency.
- Descriptor Sets are activated when the Command Buffer is filled. Different values for the uniform buffer variables can be toggled by just swapping out the Descriptor Set that points to GPU memory, rather than re-writing the GPU memory.
- Values for the shaders' uniform buffer variables can be compartmentalized into what quantities change often and what change seldom (scene-level, model-level, draw-level), so that uniform variables need to be re-written no more often than is necessary.

```
for( sporadically )
{
    Bind Descriptor Set #0
    for( the entire scene )
    {
        Bind Descriptor Set #1
        for( each object in the scene )
        {
            Bind Descriptor Set #2
            Do the drawing
        }
    }
}
```



Descriptor Sets

Our example will assume the following shader uniform variables:

```
// non-opaque must be in a uniform block:
layout( std140, set = 0, binding = 0 ) uniform sporadicBuf
{
    int        uMode;
    int        uUseLighting;
    int        uNumInstances;
} Sporadic;

layout( std140, set = 1, binding = 0 ) uniform sceneBuf
{
    mat4       uProjection;
    mat4       uView;
    mat4       uSceneOrient;
    vec4       uLightPos;
    vec4       uLightColor;
    vec4       uLightKaKdKs;
    float      uTime;
} Scene;

layout( std140, set = 2, binding = 0 ) uniform objectBuf
{
    mat4       uModel;
    mat4       uNormal;
    vec4       uColor;
    float      uShininess;
} Object;

layout( set = 3, binding = 0 ) uniform sampler2D uSampler;
```



Descriptor Sets

CPU:

GPU:

GPU:

Uniform data created in a C++ data structure

Uniform data in a "blob"*

Uniform data used in the shader

```

struct sporadicBuf
{
    int      uMode;
    int      uUseLighting;
    int      uNumInstances;
} Sporadic;

struct sceneBuf
{
    glm::mat4 uProjection;
    glm::mat4 uView;
    glm::mat4 uSceneOrient;
    glm::vec4 uLightPos;
    glm::vec4 uLightColor;
    glm::vec4 uLightKaKdKs;
    float     uTime;
} Scene;

struct objectBuf
{
    glm::mat4 uModel;
    glm::mat4 uNormal;
    glm::vec4 uColor;
    float     uShininess;
} Object;
    
```

```

1011100101010111110100010
0001011101101011101001101
1001100000011101011110011
1011010011001011111010111
0011011010100000100100011
1101000100101010101001111
1100100001110010101010011
0001101101011011111011111
1111100110101010000101110
1100110010001101000101001
1111110101110100110001111
0111010010010001101011000
0000111110001100001000001
1011001110101000111010001
1001100110010000110000110
0111100100111101001000100
1011001110001011100000010
100001011111010111110011
1000101110000110011101001
1110011111011101111111101
1111101001111011110101111
0010101000000111100100110
0111001111101001011001110
1100111000110110001111011
0000111110001110010110010
0111001101011101110010100
    
```

```

// non-opaque must be in a uniform block:
layout( std140, set = 0, binding = 0 ) uniform sporadicBuf
{
    int      uMode;
    int      uUseLighting;
    int      uNumInstances;
} Sporadic;

layout( std140, set = 1, binding = 0 ) uniform sceneBuf
{
    mat4     uProjection;
    mat4     uView;
    mat4     uSceneOrient;
    vec4     uLightPos;
    vec4     uLightColor;
    vec4     uLightKaKdKs;
    float    uTime;
} Scene;

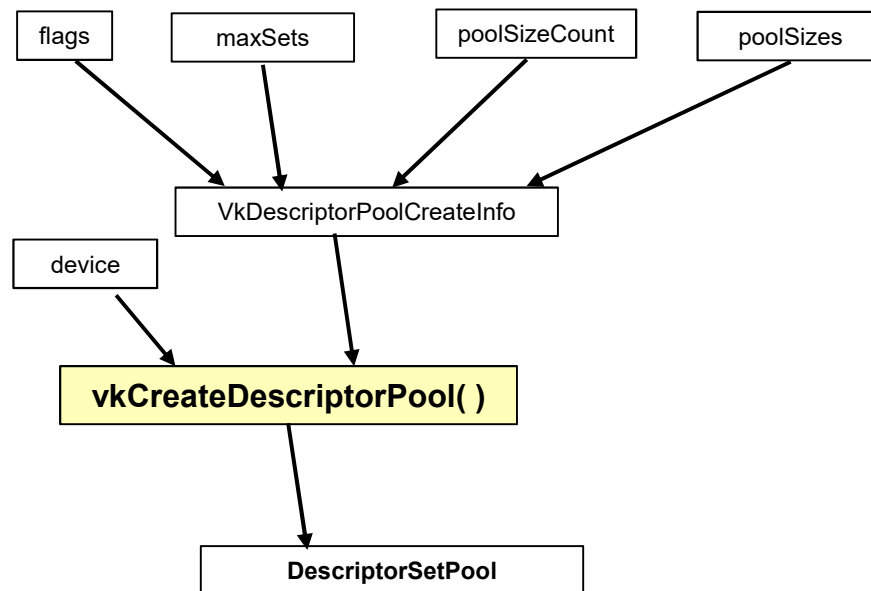
layout( std140, set = 2, binding = 0 ) uniform objectBuf
{
    mat4     uModel;
    mat4     uNormal;
    vec4     uColor;
    float    uShininess;
} Object;

layout( set = 3, binding = 0 ) uniform sampler2D uSampler;
    
```

* "binary large object"

Step 1: Descriptor Set Pools

You don't allocate Descriptor Sets on the fly – that is too slow. Instead, you allocate a “pool” of Descriptor Sets during initialization and then pull from that pool later.



```

VkResult
Init13DescriptorSetPool( )
{
    VkResult result;

    VkDescriptorPoolSize
        vdps[0].type = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
        vdps[0].descriptorCount = 1;
        vdps[1].type = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
        vdps[1].descriptorCount = 1;
        vdps[2].type = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
        vdps[2].descriptorCount = 1;
        vdps[3].type = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER;
        vdps[3].descriptorCount = 1;

    #ifdef CHOICES
    VK_DESCRIPTOR_TYPE_SAMPLER
    VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE
    VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER
    VK_DESCRIPTOR_TYPE_STORAGE_IMAGE
    VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER
    VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER
    VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER
    VK_DESCRIPTOR_TYPE_STORAGE_BUFFER
    VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC
    VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC
    VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT
    #endif

    VkDescriptorPoolCreateInfo
        vdpci.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO;
        vdpci.pNext = nullptr;
        vdpci.flags = 0;
        vdpci.maxSets = 4;
        vdpci.poolSizeCount = 4;
        vdpci.pPoolSizes = &vdps[0];

    result = vkCreateDescriptorPool( LogicalDevice, IN &vdpci, PALLOCATOR, OUT &DescriptorPool );
    return result;
}

```



Oregon State
University

Computer Graphics

Step 2: Define the Descriptor Set Layouts

I think of Descriptor Set Layouts as a kind of “Rosetta Stone” that allows the Graphics Pipeline data structure to allocate room for the uniform variables and to access them.



<https://www.britishmuseum.org>

```

// non-opaque must be in a uniform block:
layout( std140, set = 0, binding = 0 ) uniform sporadicBuf
{
    int        uMode;
    int        uUseLighting;
    int        uNumInstances;
} Sporadic;

layout( std140, set = 1, binding = 0 ) uniform sceneBuf
{
    mat4       uProjection;
    mat4       uView;
    mat4       uSceneOrient;
    vec4       uLightPos;
    vec4       uLightColor;
    vec4       uLightKaKdKs;
    float      uTime;
} Scene;

layout( std140, set = 2, binding = 0 ) uniform objectBuf
{
    mat4       uModel;
    mat4       uNormal;
    vec4       uColor;
    float      uShininess;
} Object;

layout( set = 3, binding = 0 ) uniform sampler2D uSampler;

```

SporadicSet DS Layout Binding:

binding
descriptorType
descriptorCount
pipeline stage(s)

set = 0

SceneSet DS Layout Binding:

binding
descriptorType
descriptorCount
pipeline stage(s)

set = 1

ObjectSet DS Layout Binding:

binding
descriptorType
descriptorCount
pipeline stage(s)

set = 2

TexSamplerSet DS Layout Binding:

binding
descriptorType
descriptorCount
pipeline stage(s)

set = 3




```

VkResult
Init13DescriptorSetLayouts( )
{
    VkResult result;

    //DS #0:
    VkDescriptorSetLayoutBinding      SporadicSet[1];
        SporadicSet[0].binding        = 0;
        SporadicSet[0].descriptorType  = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
        SporadicSet[0].descriptorCount = 1;
        SporadicSet[0].stageFlags      = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;
        SporadicSet[0].pImmutableSamplers = (VkSampler *)nullptr;

    // DS #1:
    VkDescriptorSetLayoutBinding      SceneSet[1];
        SceneSet[0].binding            = 0;
        SceneSet[0].descriptorType     = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
        SceneSet[0].descriptorCount    = 1;
        SceneSet[0].stageFlags         = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;
        SceneSet[0].pImmutableSamplers = (VkSampler *)nullptr;

    //DS #2:
    VkDescriptorSetLayoutBinding      ObjectSet[1];
        ObjectSet[0].binding           = 0;
        ObjectSet[0].descriptorType    = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
        ObjectSet[0].descriptorCount   = 1;
        ObjectSet[0].stageFlags        = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;
        ObjectSet[0].pImmutableSamplers = (VkSampler *)nullptr;

    // DS #3:
    VkDescriptorSetLayoutBinding      TexSamplerSet[1];
        TexSamplerSet[0].binding       = 0;
        TexSamplerSet[0].descriptorType = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER;
        TexSamplerSet[0].descriptorCount = 1;
        TexSamplerSet[0].stageFlags     = VK_SHADER_STAGE_FRAGMENT_BIT;
        TexSamplerSet[0].pImmutableSamplers = (VkSampler *)nullptr;

```

```

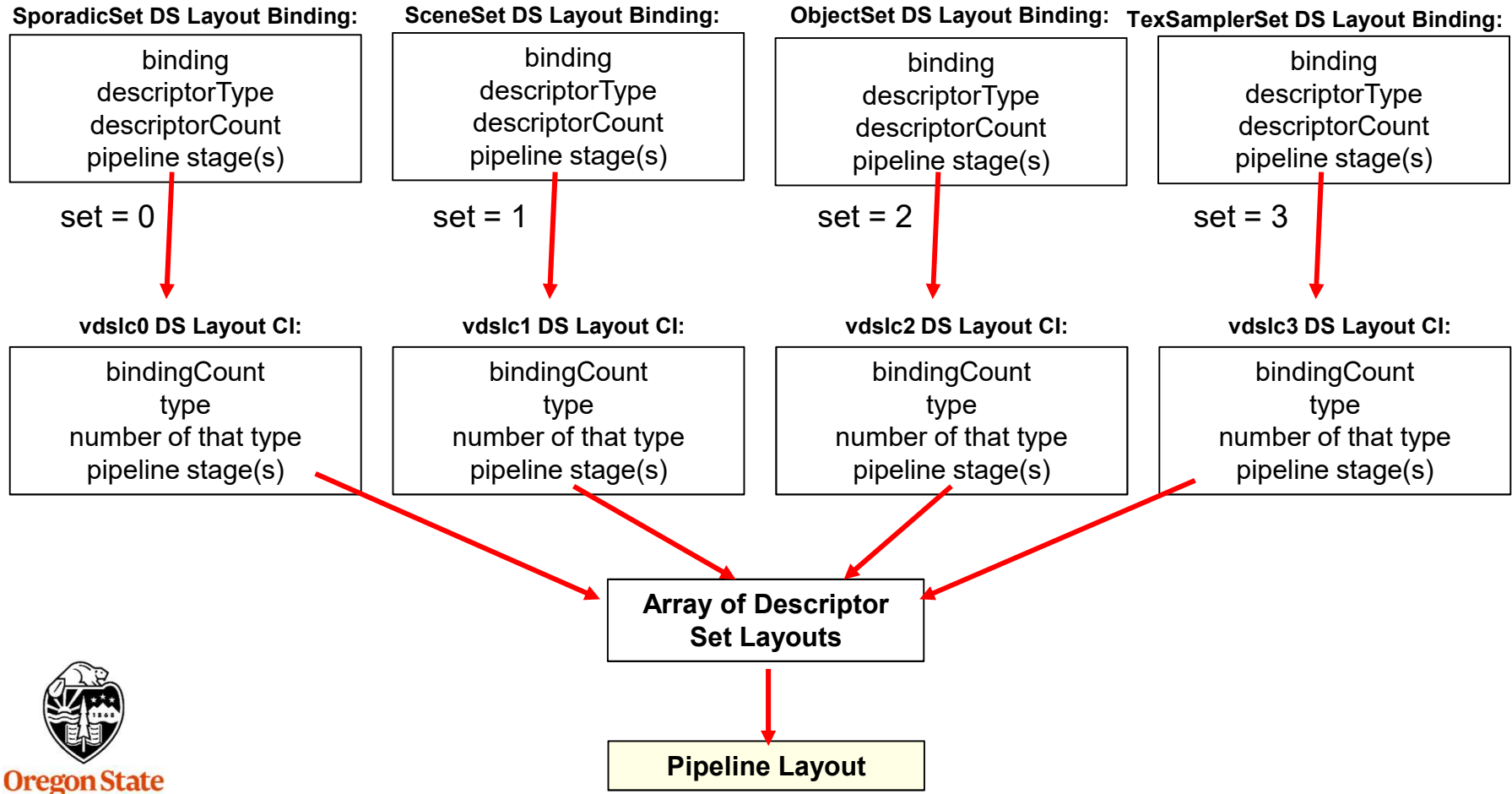
uniform sampler2D uSampler;
vec4 rgba = texture( uSampler, vST );

```



Step 2: Define the Descriptor Set Layouts

```
// globals:
VkDescriptorPool          DescriptorPool;
VkDescriptorSetLayout     DescriptorSetLayouts[4];
VkDescriptorSet           DescriptorSets[4];
```



```

VkDescriptorSetLayoutCreateInfo vdslc0;
    vdslc0.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
    vdslc0.pNext = nullptr;
    vdslc0.flags = 0;
    vdslc0.bindingCount = 1;
    vdslc0.pBindings = &SporadicSet[0];

VkDescriptorSetLayoutCreateInfo vdslc1;
    vdslc1.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
    vdslc1.pNext = nullptr;
    vdslc1.flags = 0;
    vdslc1.bindingCount = 1;
    vdslc1.pBindings = &SceneSet[0];

VkDescriptorSetLayoutCreateInfo vdslc2;
    vdslc2.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
    vdslc2.pNext = nullptr;
    vdslc2.flags = 0;
    vdslc2.bindingCount = 1;
    vdslc2.pBindings = &ObjectSet[0];

VkDescriptorSetLayoutCreateInfo vdslc3;
    vdslc3.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
    vdslc3.pNext = nullptr;
    vdslc3.flags = 0;
    vdslc3.bindingCount = 1;
    vdslc3.pBindings = &TexSamplerSet[0];

result = vkCreateDescriptorSetLayout( LogicalDevice, IN &vdslc0, PALLOCATOR, OUT &DescriptorSetLayouts[0] );
result = vkCreateDescriptorSetLayout( LogicalDevice, IN &vdslc1, FALLOCATOR, OUT &DescriptorSetLayouts[1] );
result = vkCreateDescriptorSetLayout( LogicalDevice, IN &vdslc2, FALLOCATOR, OUT &DescriptorSetLayouts[2] );
result = vkCreateDescriptorSetLayout( LogicalDevice, IN &vdslc3, PALLOCATOR, OUT &DescriptorSetLayouts[3] );

return result;
}

```

Step 3: Include the Descriptor Set Layouts in a Graphics Pipeline Layout

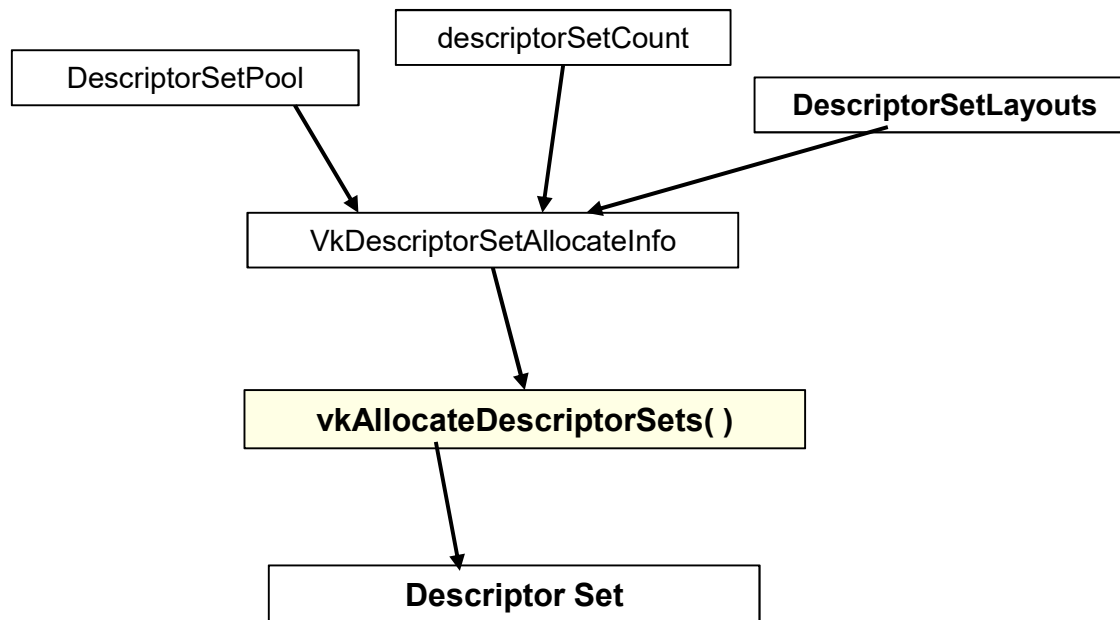
12

```
VkResult  
Init14GraphicsPipelineLayout( )  
{  
    VkResult result;  
  
    VkPipelineLayoutCreateInfo  
        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;  
}
```



Step 4: Allocating the Memory for Descriptor Sets

13



Step 4: Allocating the Memory for Descriptor Sets

14

```
VkResult  
Init13DescriptorSets( )  
{  
    VkResult result;  
  
    VkDescriptorSetAllocateInfo vdsai;  
    vdsai.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO;  
    vdsai.pNext = nullptr;  
    vdsai.descriptorPool = DescriptorPool;  
    vdsai.descriptorSetCount = 4;  
    vdsai.pSetLayouts = DescriptorSetLayouts;  
  
    result = vkAllocateDescriptorSets( LogicalDevice, IN &vdsai, OUT &DescriptorSets[0] );
```

Step 5: Tell the Descriptor Sets where their CPU Data is

```
VkDescriptorBufferInfo      vdbi0;  
    vdbi0.buffer = MySporadicUniformBuffer.buffer;  
    vdbi0.offset = 0;  
    vdbi0.range = sizeof(Sporadic);
```

This struct identifies what buffer it owns and how big it is

```
VkDescriptorBufferInfo      vdbi1;  
    vdbi1.buffer = MySceneUniformBuffer.buffer;  
    vdbi1.offset = 0;  
    vdbi1.range = sizeof(Scene);
```

This struct identifies what buffer it owns and how big it is

```
VkDescriptorBufferInfo      vdbi2;  
    vdbi2.buffer = MyObjectUniformBuffer.buffer;  
    vdbi2.offset = 0;  
    vdbi2.range = sizeof(Object);
```

This struct identifies what buffer it owns and how big it is

```
VkDescriptorImageInfo       vdii0;  
    vdii.sampler = MyPuppyTexture.texSampler;  
    vdii.imageView = MyPuppyTexture.texImageView;  
    vdii.imageLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
```

This struct identifies what texture sampler and image view it owns



Step 5: Tell the Descriptor Sets where their CPU Data is

```
VkWriteDescriptorSet          vwds0;
// ds 0:
vwds0.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
vwds0.pNext = nullptr;
vwds0.dstSet = DescriptorSets[0];
vwds0.dstBinding = 0;
vwds0.dstArrayElement = 0;
vwds0.descriptorCount = 1;
vwds0.descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
vwds0.pBufferInfo = IN &vdbi0;
vwds0.pImageInfo = (VkDescriptorImageInfo *)nullptr;
vwds0.pTexelBufferView = (VkBufferView *)nullptr;
```

This struct links a Descriptor Set to the buffer it is pointing to

```
// ds 1:
VkWriteDescriptorSet          vwds1;
vwds1.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
vwds1.pNext = nullptr;
vwds1.dstSet = DescriptorSets[1];
vwds1.dstBinding = 0;
vwds1.dstArrayElement = 0;
vwds1.descriptorCount = 1;
vwds1.descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
vwds1.pBufferInfo = IN &vdbi1;
vwds1.pImageInfo = (VkDescriptorImageInfo *)nullptr;
vwds1.pTexelBufferView = (VkBufferView *)nullptr;
```

This struct links a Descriptor Set to the buffer it is pointing to



Step 5: Tell the Descriptor Sets where their data is

```
VkWriteDescriptorSet          vwds2;
// ds 2:
vwds2.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
vwds2.pNext = nullptr;
vwds2.dstSet = DescriptorSets[2];
vwds2.dstBinding = 0;
vwds2.dstArrayElement = 0;
vwds2.descriptorCount = 1;
vwds2.descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
vwds2.pBufferInfo = IN &vdbi2;
vwds2.pImageInfo = (VkDescriptorImageInfo *)nullptr;
vwds2.pTexelBufferView = (VkBufferView *)nullptr;
```

This struct links a Descriptor Set to the buffer it is pointing to

```
// ds 3:
VkWriteDescriptorSet          vwds3;
vwds3.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
vwds3.pNext = nullptr;
vwds3.dstSet = DescriptorSets[3];
vwds3.dstBinding = 0;
vwds3.dstArrayElement = 0;
vwds3.descriptorCount = 1;
vwds3.descriptorType = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER;
vwds3.pBufferInfo = (VkDescriptorBufferInfo *)nullptr;
vwds3.pImageInfo = IN &vdii0;
vwds3.pTexelBufferView = (VkBufferView *)nullptr;
```

This struct links a Descriptor Set to the image it is pointing to

```
uint32_t copyCount = 0;
```

```
// this could have been done with one call and an array of VkWriteDescriptorSets:
```

```
vkUpdateDescriptorSets( LogicalDevice, 1, IN &vwds0, IN copyCount, (VkCopyDescriptorSet *)nullptr );
vkUpdateDescriptorSets( LogicalDevice, 1, IN &vwds1, IN copyCount, (VkCopyDescriptorSet *)nullptr );
vkUpdateDescriptorSets( LogicalDevice, 1, IN &vwds2, IN copyCount, (VkCopyDescriptorSet *)nullptr );
vkUpdateDescriptorSets( LogicalDevice, 1, IN &vwds3, IN copyCount, (VkCopyDescriptorSet *)nullptr );
```



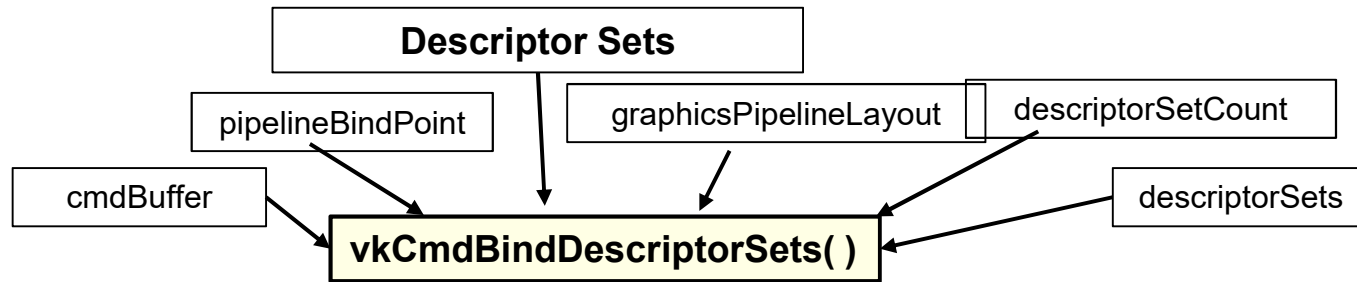
Step 6: Include the Descriptor Set Layout when Creating a Graphics Pipeline 18

```
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 = &vrprsci;
    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 &GraphicsPipeline );
```



Step 7: Bind Descriptor Sets into the Command Buffer when Drawing



```
vkCmdBindDescriptorSets( CommandBuffers[nextImageIndex],  
VK_PIPELINE_BIND_POINT_GRAPHICS, GraphicsPipelineLayout,  
0, 4, DescriptorSets, 0, (uint32_t *)nullptr );
```

So, the Pipeline Layout contains the **structure** of the Descriptor Sets.
Any collection of Descriptor Sets that match that structure can be bound into that pipeline.

Sidebar: The Entire Descriptor Set Journey

VkDescriptorPoolCreateInfo
vkCreateDescriptorPool()

Create the pool of Descriptor Sets for future use

VkDescriptorSetLayoutBinding
VkDescriptorSetLayoutCreateInfo
vkCreateDescriptorSetLayout()
vkCreatePipelineLayout()

Describe a particular Descriptor Set layout and use it in a specific Pipeline layout

VkDescriptorSetAllocateInfo
vkAllocateDescriptorSets()

Allocate memory for particular Descriptor Sets

VkDescriptorBufferInfo
VkDescriptorImageInfo
VkWriteDescriptorSet
vkUpdateDescriptorSets()

Tell a particular Descriptor Set where its CPU data is

Re-write CPU data into a particular Descriptor Set

vkCmdBindDescriptorSets()

Make a particular Descriptor Set “current” for rendering

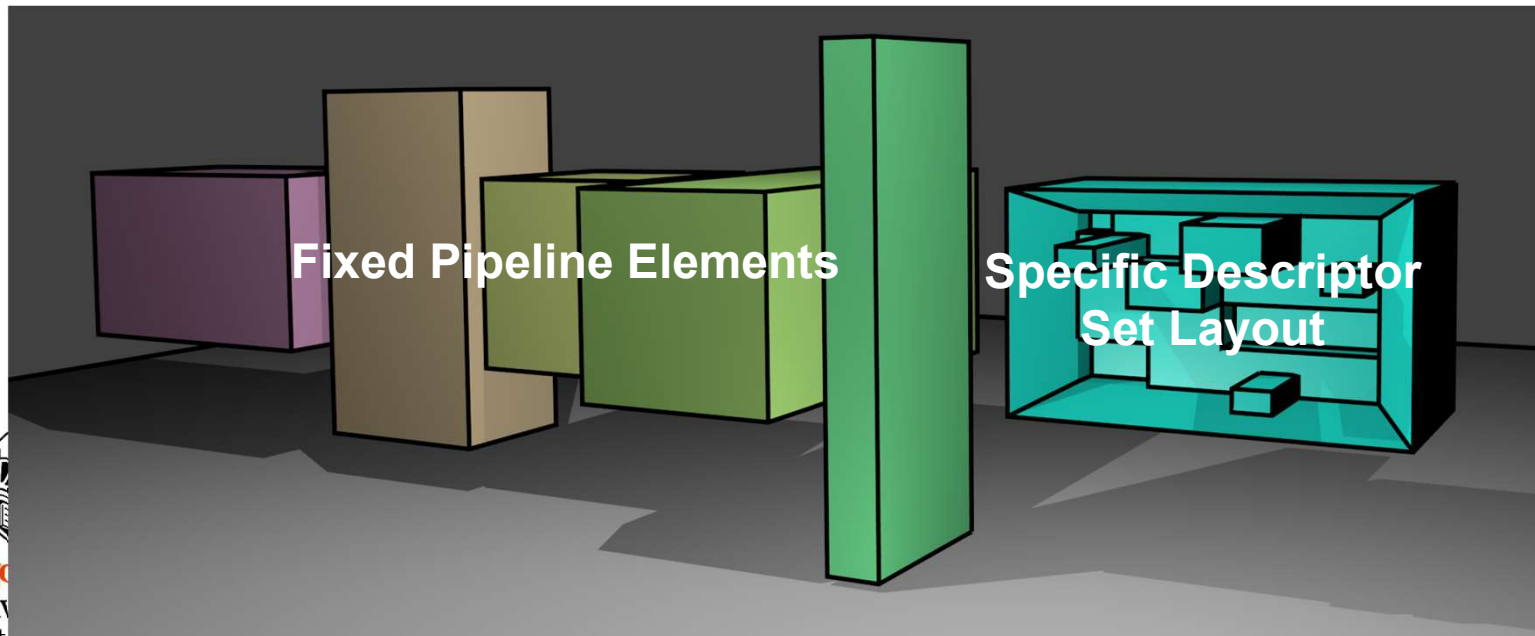


Sidebar: Why Do Descriptor Sets Need to Provide Layout Information to the Pipeline Data Structure?

The pieces of the Pipeline Data Structure are fixed in size – with the exception of the Descriptor Sets and the Push Constants. Each of these two can be any size, depending on what you allocate for them. So, the Pipeline Data Structure needs to know how these two are configured before it can set its own total layout.

Think of the DS layout as being a particular-sized hole in the Pipeline Data Structure. Any data you have that matches this hole's shape and size can be plugged in there.

The Pipeline Data Structure



Sidebar: Why Do Descriptor Sets Need to Provide Layout Information to the Pipeline Data Structure?

Any set of data that matches the Descriptor Set Layout can be plugged in there.

