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?

```glsl
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.
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.

```cpp
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
        }
    }
}
```

What are Descriptor Sets?

Descriptor Sets

Our example will assume the following shader uniform variables:

```cpp
// 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;
```
Uniform data created in a C++ data structure

CPU:

Uniform data in a “blob”*

GPU:

Uniform data used in the shader

C++

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;

* “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.
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.

<table>
<thead>
<tr>
<th>Binding</th>
<th>Descriptor Type</th>
<th>Descriptor Count</th>
<th>Pipeline Stage(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>uniform</td>
<td>4</td>
<td>fragment</td>
</tr>
<tr>
<td>1</td>
<td>image</td>
<td>1</td>
<td>fragment</td>
</tr>
<tr>
<td>2</td>
<td>image</td>
<td>1</td>
<td>fragment</td>
</tr>
<tr>
<td>3</td>
<td>image</td>
<td>1</td>
<td>fragment</td>
</tr>
</tbody>
</table>

SporadicSet DS Layout Binding:
- binding = 0
- descriptorType: uniform
- descriptorCount: 4
- pipeline stage(s): fragment

SceneSet DS Layout Binding:
- binding = 1
- descriptorType: image
- descriptorCount: 1
- pipeline stage(s): fragment

ObjectSet DS Layout Binding:
- binding = 2
- descriptorType: image
- descriptorCount: 1
- pipeline stage(s): fragment

TexSamplerSet DS Layout Binding:
- binding = 3
- descriptorType: image
- descriptorCount: 1
- pipeline stage(s): fragment
Step 2: Define the Descriptor Set Layouts

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

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

// Init Descriptor Set Layouts

VkResult Init13DescriptorSetLayouts()
{
    // globals:
    VkDescriptorPool DescriptorPool;
    VkDescriptorSetLayout DescriptorSetLayouts[4];
    VkDescriptorSet DescriptorSets[4];

    VkResult result = VK_SUCCESS;

    // Sporadic Set Layout Binding
    SporadicSet[1].binding = 0;
    SporadicSet[1].descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
    SporadicSet[1].descriptorCount = 1;
    SporadicSet[1].stageFlag = (VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT);
    SporadicSet[1].pImmutableSamplers = (&uSampler);

    // Scene Set Layout Binding
    SceneSet[1].binding = 0;
    SceneSet[1].descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
    SceneSet[1].descriptorCount = 1;
    SceneSet[1].stageFlag = (VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT);
    SceneSet[1].pImmutableSamplers = (VkSampler *)((char *)nullptr);

    // Object Set Layout Binding
    ObjectSet[1].binding = 0;
    ObjectSet[1].descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
    ObjectSet[1].descriptorCount = 1;
    ObjectSet[1].stageFlag = (VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT);
    ObjectSet[1].pImmutableSamplers = (VkSampler *)((char *)nullptr);

    // Tex Sampler Set Layout Binding
    TexSamplerSet[1].binding = 0;
    TexSamplerSet[1].descriptorType = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER;
    TexSamplerSet[1].descriptorCount = 1;
    TexSamplerSet[1].stageFlag = VK_SHADER_STAGE_FRAGMENT_BIT;
    TexSamplerSet[1].pImmutableSamplers = (VkSampler *)((char *)nullptr);

    return result;
}
Step 3: Include the Descriptor Set Layouts in a Graphics Pipeline Layout

```c
int14GraphicsPipelineLayout( )
{
    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, &vplci, PALLOCATOR, &GraphicsPipelineLayout );

    return result;
}
```

University
Computer Graphics

mjb – January 5, 2023

1/5/2023
Step 4: Allocating the Memory for Descriptor Sets

```
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

```
// ds 0:
VkWriteDescriptorSet vwds0;
    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 = &vdbi0;
    vwds0.pImageInfo = (VkDescriptorImageInfo *)nullptr;
    vwds0.pTexelBufferView = (VkBufferView *)nullptr;

// 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 = &vdbi1;
    vwds1.pImageInfo = (VkDescriptorImageInfo *)nullptr;
    vwds1.pTexelBufferView = (VkBufferView *)nullptr;
```

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

```
// ds 2:
VkWriteDescriptorSet vwds2;
    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 = &vdbi2;
    vwds2.pImageInfo = (VkDescriptorImageInfo *)nullptr;
    vwds2.pTexelBufferView = (VkBufferView *)nullptr;

// 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_UNIFORM_BUFFER;
    vwds3.pBufferInfo = (VkBufferView *)nullptr;
    vwds3.pImageInfo = (VkDescriptorImageInfo *)nullptr;
```

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

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

This struct identifies what texture sampler and image view it owns.
Step 5: Tell the Descriptor Sets where their data is

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

```c
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 image it is pointing to:

```c
VkWriteDescriptorSet vwds3;
// ds 3:
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;
```

uint32_t copyCount = 0;

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

```c
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

This struct links a Graphics Pipeline Layout:

```c
VkGraphicsPipelineCreateInfo vgpci;
vgoci.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
vgoci.pNext = nullptr;
vgoci.flags = 0;

#ifdef CHOICES
VK_PIPELINE_CREATE_DISABLE_OPTIMIZATION_BIT
VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT
VK_PIPELINE_CREATE_DERIVATIVE_BIT
#endif
vgoci.stageCount = 2;                           // number of stages in this pipeline
vgoci.pStages = vpsci;
vgoci.pVertexInputState = &vpvisci;
vgoci.pInputAssemblyState = &vpiasci;
vgoci.pTessellationState = (VkPipelineTessellationStateCreateInfo *)nullptr;
vgoci.pViewportState = &vpvsci;
vgoci.pRasterizationState = &vprsci;
vgoci.pMultisampleState = &vpmsci;
vgoci.pDepthStencilState = &vpdsci;
vgoci.pColorBlendState = &vpcbsci;
vgoci.pDynamicState = &vpdsci;
vgoci.layout = IN GraphicsPipelineLayout;
vgoci.renderPass = IN RenderPass;               // subpass number
vgoci.subpass = 0;
vgoci.basePipelineHandle = (VkPipeline) VK_NULL_HANDLE;
vgoci.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,
4, DescriptorSets, (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( ) \rightarrow \text{Create the pool of Descriptor Sets for future use}

VkDescriptorSetLayoutBinding
VkDescriptorSetLayoutCreateInfo
vkCreateDescriptorsetLayout( )
vkCreatePipelineLayout( ) \rightarrow \text{Describe a particular Descriptor Set layout and use it in a specific Pipeline layout}

VkDescriptorSetAllocateInfo
vkAllocateDescriptorSets( ) \rightarrow \text{Allocate memory for particular Descriptor Sets}

VkWriteDescriptorSet
VkDescriptorBufferInfo
VkDescriptorImageInfo
VkWriteDescriptorSet
vkUpdateDescriptorSets( ) \rightarrow \text{Re-write CPU data into a particular Descriptor Set}

vkCmdBindDescriptorSets( ) \rightarrow \text{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

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