**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:

1. Related uniform variables can be updated as a group, gaining efficiency.
2. 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.
3. Values for the shaders' uniform buffer variables can be compartmentalized into what quantities change often and what change seldom (scene-level, model-level, state-level), so that uniform variables need to be re-written no more often than is necessary.

In OpenGL, these are all in one set. They all get bound, whether you need them here or not.

**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 bunch of related uniform variables all at once?

```
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(std140, binding = 7) uniform sampler2D uSampler;
```

**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 and then pull from that pool later.

You don’t allocate Descriptor Sets on the fly — that is too slow. Instead, you allocate a “pool” of Descriptor Sets and then pull from that pool later.
### Step 2: Define the Descriptor Set Layouts

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

#### Matrix Set DS Layout Binding
- **LightSet DS Layout Binding**
- **MiscSet DS Layout Binding**
- **TexSamplerSet DS Layout Binding**

#### Matrix Set DS Layout CI
- **descriptorCount**
- **pipeline stage(s)**
- **descriptorType**
- **binding**

#### LightSet DS Layout CI
- **descriptorCount**
- **pipeline stage(s)**
- **descriptorType**
- **binding**

#### MiscSet DS Layout CI
- **descriptorCount**
- **pipeline stage(s)**
- **descriptorType**
- **binding**

#### TexSamplerSet DS Layout CI
- **descriptorCount**
- **pipeline stage(s)**
- **descriptorType**
- **binding**

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

```cpp
VkResult vplci = VkPipelineLayoutCreateInfo
{ .sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO, .setLayoutCount = 4, .pSetLayouts = &DescriptorSetLayouts[0], .pNext = nullptr, .flags = 0 }; VkResult result;
```

---

**Pipeline data structure to allocate room for the uniform variables and to access them.**
Step 4: Allocating the Memory for Descriptor Sets

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

// ds 0:
vwds0.pTexelBufferView = (VkBufferView *)nullptr;
vwds0.pImageInfo = (VkDescriptorImageInfo *)nullptr;
vwds0.pBufferInfo = (VkDescriptorBufferInfo *)nullptr;
vwds0.descriptorCount = 1;
vwds0.dstArrayElement = 0;
vwds0.dstBinding = 0;
vwds0.pNext = nullptr;
vwds0.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
// ds 1:
vwds1.pTexelBufferView = (VkBufferView *)nullptr;
vwds1.pBufferInfo = (VkDescriptorBufferInfo *)nullptr;
vwds1.descriptorCount = 1;
vwds1.dstArrayElement = 0;
vwds1.dstBinding = 0;
vwds1.pNext = nullptr;
vwds1.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
// ds 2:
vwds2.pTexelBufferView = (VkBufferView *)nullptr;
vwds2.pBufferInfo = (VkDescriptorBufferInfo *)nullptr;
vwds2.descriptorCount = 1;
vwds2.dstArrayElement = 0;
vwds2.dstBinding = 0;
vwds2.pNext = nullptr;
vwds2.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
```

Step 5: Tell the Descriptor Sets where their data is

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

```cpp
// ds 0:
vdii.imageLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
vdii.sampler = MyPuppyTexture.texSampler;
```

```cpp
vdbi2.range = sizeof(Misc);
vdbi2.offset = 0;
vdbi2.buffer = MyMiscUniformBuffer.buffer;
```

```cpp
vdbi1.range = sizeof(Light);
vdbi1.offset = 0;
vdbi1.buffer = MyLightUniformBuffer.buffer;
```

```cpp
vdbi0.range = sizeof(Matrices);
vdbi0.buffer = MyMatrixUniformBuffer.buffer;
```

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

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

// ds 0:
vwds0.pTexelBufferView = (VkBufferView *)nullptr;
vwds0.pImageInfo = (VkDescriptorImageInfo *)nullptr;
vwds0.pBufferInfo = (VkDescriptorBufferInfo *)nullptr;
vwds0.descriptorCount = 1;
vwds0.dstArrayElement = 0;
vwds0.dstBinding = 0;
vwds0.pNext = nullptr;
vwds0.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
// ds 1:
vwds1.pTexelBufferView = (VkBufferView *)nullptr;
vwds1.pBufferInfo = (VkDescriptorBufferInfo *)nullptr;
vwds1.descriptorCount = 1;
vwds1.dstArrayElement = 0;
vwds1.dstBinding = 0;
vwds1.pNext = nullptr;
vwds1.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
// ds 2:
vwds2.pTexelBufferView = (VkBufferView *)nullptr;
vwds2.pBufferInfo = (VkDescriptorBufferInfo *)nullptr;
vwds2.descriptorCount = 1;
vwds2.dstArrayElement = 0;
vwds2.dstBinding = 0;
vwds2.pNext = nullptr;
vwds2.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
```

// ds 0:
vdii.imageLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
vdii.sampler = MyPuppyTexture.texSampler;
```

```cpp
vdbi2.range = sizeof(Misc);
vdbi2.offset = 0;
vdbi2.buffer = MyMiscUniformBuffer.buffer;
```

```cpp
vdbi1.range = sizeof(Light);
vdbi1.offset = 0;
vdbi1.buffer = MyLightUniformBuffer.buffer;
```

```cpp
vdbi0.range = sizeof(Matrices);
vdbi0.buffer = MyMatrixUniformBuffer.buffer;
```
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 );