### 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 shader's 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.**

### In OpenGl

OpenGL uses 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?

### Descriptor Sets

Our example will assume the following shader uniform variables:

```glsl
layout( std140, set = 0, binding = 0 ) uniform mat4 uModelMatrix;
layout( std140, set = 1, binding = 0 ) uniform mat4 uViewMatrix;
layout( std140, set = 2, binding = 0 ) uniform mat4 uProjectionMatrix;
layout( std140, set = 3, binding = 0 ) uniform sampler2D uSampler;
```

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?

### 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.
### Step 1: 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.

#### Matrix Set DS Layout Binding:
- **pipeline stage(s):** `pipelineStageFlags = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT`
- **descriptorCount:** `1`
- **descriptorType:** `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`
- **binding:** `set = 0`

#### Light Set DS Layout Binding:
- **pipeline stage(s):** `pipelineStageFlags = VK_SHADER_STAGE_FRAGMENT_BIT`
- **descriptorCount:** `1`
- **descriptorType:** `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER`
- **binding:** `set = 1`

#### Misc Set DS Layout Binding:
- **pipeline stage(s):** `pipelineStageFlags = VK_SHADER_STAGE_FRAGMENT_BIT`
- **descriptorCount:** `1`
- **descriptorType:** `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`
- **binding:** `set = 2`

#### TexSampler Set DS Layout Binding:
- **pipeline stage(s):** `pipelineStageFlags = VK_SHADER_STAGE_FRAGMENT_BIT`
- **descriptorCount:** `1`
- **descriptorType:** `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`
- **binding:** `set = 3`

### Step 2: Include the Descriptor Set Layouts in a Graphics Pipeline Layout

```c
VkResult vplci = vkCreatePipelineLayout( LogicalDevice, IN &vdslc0, PALLOCATOR, OUT &GraphicsPipelineLayout );
```

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

```c
VkResult vplci = vkCreatePipelineLayout( LogicalDevice, IN &vdslc1, PALLOCATOR, OUT &GraphicsPipelineLayout );
```

```c
VkResult vplci = vkCreatePipelineLayout( LogicalDevice, IN &vdslc2, PALLOCATOR, OUT &GraphicsPipelineLayout );
```

```c
VkResult vplci = vkCreatePipelineLayout( LogicalDevice, IN &vdslc3, PALLOCATOR, OUT &GraphicsPipelineLayout );
```
### Step 4: Allocating the Memory for Descriptor Sets

```c
// Allocating the Memory for Descriptor Sets

VkDescriptorPool vdp;  // Used to allocate the Descriptor Pool

vkAllocateDescriptorSets( LogicalDevice, 1, IN &vdsai );

DescriptorSets[0] = &vwds0;
DescriptorSets[1] = &vwds1;
DescriptorSets[2] = &vwds2;
DescriptorSets[3] = &vwds3;
```

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

```c
// This struct links a Descriptor Set to the buffer it is pointing to
VkDescriptorSet vds;

vds.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET;
```

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

```c
// Include the Descriptor Set Layout when Creating a Graphics Pipeline

VkGraphicsPipelineCreateInfo vgpci;

vgpci.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
vgpci.layout = IN vdsai.pSetLayouts[0];
vgpci.stageCount = 2;  // number of stages in this pipeline
```

### Diagrams

1. **Diagram 1:** Illustrates the process of creating a Descriptor Pool and allocating Descriptor Sets.
2. **Diagram 2:** Shows the relationship between Descriptor Sets and their respective pools.
3. **Diagram 3:** Demonstrates the linking of Descriptor Sets to their respective buffer views.
4. **Diagram 4:** Represents the inclusion of a Descriptor Set Layout in the creation of a graphics pipeline.
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.