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.

Here’s our example from OpenGL:

```glsl
layout( set = 3, binding = 0 ) uniform sampler2D uSampler;
layout( std140, set = 2, binding = 0 ) uniform miscBuf
{
  layout( std140, set = 1, binding = 0 ) uniform lightBuf
  {
    layout( std140, set = 0, binding = 0 ) uniform matBuf
    {
      layout( std140, set = 0, binding = 0 ) uniform mat4 uModelMatrix;
      layout( std140, set = 0, binding = 0 ) uniform mat4 uViewMatrix;
      layout( std140, set = 0, binding = 0 ) uniform mat4 uProjectionMatrix;
      layout( std140, set = 0, binding = 0 ) uniform vec4 uLightPos;
      layout( std140, binding = 7 ) uniform sampler2D uSampler;
    }
  }
}
```

Wouldn’t it be nice if we could update a bunch of related uniform variables all at once?

Each uniform variable gets updated one-at-a-time. You can get at each one.

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

Wouldn’t it be nice if we could update a bunch of related uniform variables all at once?

In OpenGL

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?

For each scene:

- Bind Descriptor Set #0
- Do the drawing

For each object:

- Bind Descriptor Set #1
- Do the drawing

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.
**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.

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

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

```c
result = vkAllocateDescriptorSets( LogicalDevice, IN &vdsai, OUT &DescriptorSets[0] );
```

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

```c
// ds 1:
vwds0.pTexelBufferView = (VkBufferView *)nullptr;
vwds0.pImageInfo = (VkDescriptorImageInfo *)nullptr;
vwds0.pBufferInfo = IN
vwds0.descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
vwds0.descriptorCount = 1;
vwds0.dstSet = ...
vwds0.pNext = nullptr;
```

Step 5: Tell the Descriptor Sets where their data is

```c
// ds 3:
vdbi2.range = sizeof(Misc);
vdbi2.buffer = MyMiscUniformBuffer.buffer;
```

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

```c
vkCreateGraphicsPipelines(device, VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT | VK_PIPELINE_CREATE_DERIVATIVE_BIT, 0, IN &vgpci, OUT &pPipeline);
```

Additional Diagrams:

- Diagram of a VkWriteDescriptorSet structure
- Diagram of a VkDescriptorSetAllocateInfo structure
- Diagram of a VkGraphicsPipelineCreateInfo structure

Textual Content:

- Step 4: Allocating the Memory for Descriptor Sets
- Step 5: Tell the Descriptor Sets where their CPU Data is
- Step 5: Tell the Descriptor Sets where their data is
- Step 6: Include the Descriptor Set Layout when Creating 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);