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?

```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;
```

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

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, draw-level), so that uniform variables need to be re-written no more often than is necessary.

What are Descriptor Sets?

```glsl
// non-opaque must be in a uniform block:
layout( std140, set = 0, binding = 0 ) uniform matBuf
{
    mat4 uModelMatrix;
    mat4 uViewMatrix;
    mat4 uProjectionMatrix;
    mat3 uNormalMatrix;
} Matrices;

layout( std140, set = 1, binding = 0 ) uniform lightBuf
{
    float  uKa, uKd, uKs, uShininess;
    vec4 uLightPos;
    vec4 uLightSpecularColor;
    vec4 uEyePos;
} Light;

layout( std140, set = 2, binding = 0 ) uniform miscBuf
{
    float uTime;
    int uMode;
    int uLighting;
} Misc;

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

Descriptor Sets

Our example will assume the following shader uniform variables:
**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.

---

**CPU:**
- Uniform data created in a C++ data structure
  - Knows the CPU data structure
  - Knows where the data starts

**GPU:**
- Uniform data used in the shader
  - Knows the shader data structure
  - Knows where the data starts
  - Knows the data's size
  - Doesn't know the CPU or GPU data structure

---

### Descriptor Sets

**CPU:**
- Uniform data created in a C++ data structure
  - Knows the CPU data structure
  - Knows where the data starts

**GPU:**
- Uniform data used in the shader
  - Knows the shader data structure
  - Knows where the data starts
  - Knows the data's size
  - Doesn't know the CPU or GPU data structure

---

### 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.

```c
VkDescriptorSetLayoutInit13DescriptorSetPool( )
```

---

**LightSet DS Layout Binding:**
- `binding: 8`
- `pipeline stage(s): 1: vertex; 2: fragment`

**Matrices Set DS Layout Binding:**
- `binding: 6`
- `pipeline stage(s): 0: all`

---

**Init13DescriptorSetPool( )**

```c
VkResult result;
result = vkCreateDescriptorPool( LogicalDevice, IN &vdpci, PALLOCATOR, OUT &DescriptorPool);
return result;
```
Step 2: Define the Descriptor Set Layouts

Matrix Set DS Layout Binding: LightSet DS Layout Binding: MiscSet DS Layout Binding: TexSamplerSet DS Layout Binding:

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

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

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

// DS #3:
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;

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

VkResult
Init14GraphicsPipelineLayout() {
VkResult result;
result = vkCreatePipelineLayout(LogicalDevice, &vplci, PALLOCATOR, OUT &GraphicsPipelineLayout);
return result;
}

Array of Descriptor Set Layouts
Step 4: Allocating the Memory for Descriptor Sets

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

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

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

```c
// 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 = IN &vdbi0
    vdbi0.buffer = MyMatrixUniformBuffer.buffer;
vdbi0.offset = 0;
vdbi0.range = sizeof(Matrices);
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 = IN &vdbi1
    vdbi1.buffer = MyLightUniformBuffer.buffer;
vdbi1.offset = 0;
vdbi1.range = sizeof(Light);
vwds1.pImageInfo = (VkDescriptorImageInfo *)nullptr;
vwds1.pTexelBufferView = (VkBufferView *)nullptr;
```

Good to use `sizeof`
Step 5: Tell the Descriptor Sets where their data is

This struct links a Descriptor Set to the image 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 = (VkBufferInfo *)nullptr;
vwds2.pImageInfo = (VkDescriptorImageInfo *)nullptr;
vwds2.pTexelBufferView = (VkBufferView *)nullptr;
```

// ds 3:

```c
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.pImageInfo = (VkDescriptorImageInfo *)nullptr;
vwds3.pTexelBufferView = (VkBufferView *)nullptr;
```

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

```c
VkGraphicsPipelineCreateInfo vgpici;
vgpici.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
vgpici.pNext = nullptr;
vgpici.flags = 0;
#endif
vgpici.stageCount = 2;  // number of stages in this pipeline = vertex + fragment
vgpici.pStages = vpssci;
vgpici.pVertexInputState = &vpvisci;
vgpici.pInputAssemblyState = &vpiasci;
vgpici.pTessellationState = (VkPipelineTessellationStateCreateInfo *)nullptr;
vgpici.pViewportState = &vpvsci;
vgpici.pRasterizationState = &vprsci;
vgpici.pMultisampleState = &vpmsci;
vgpici.pDepthStencilState = &vpdssci;
vgpici.pColorBlendState = &vpcbsci;
vgpici.pDynamicState = &vpdsci;
vgpici.layout = IN GraphicsPipelineLayout;
vgpici.renderPass = IN RenderPass;  // subpass number
vgpici.basePipelineHandle = (VkPipeline) VK_NULL_HANDLE;
vgpici.basePipelineIndex = 0;
result = vkCreateGraphicsPipelines;
```

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

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