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;
```
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.

```
for( each scene )
{
    Bind Descriptor Set #0
    for( each object )
    {
        Bind Descriptor Set #1
        for( each draw )
        {
            Bind Descriptor Set #2
            Do the drawing
        }
    }
}
```

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 matBuf
{
    mat4 uModelMatrix;
    mat4 uViewMatrix;
    mat4 uProjectionMatrix;
    mat3 uNormalMatrix;
} Matrices;

layout( std140, set = 1, binding = 0 ) uniform lightBuf
{
    vec4 uLightPos;
} Light;

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

layout( set = 3, binding = 0 ) 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.

```cpp
struct matBuf {
    glm::mat4 uModelMatrix;
    glm::mat4 uViewMatrix;
    glm::mat4 uProjectionMatrix;
    glm::mat3 uNormalMatrix;
};

struct lightBuf {
    glm::vec4 uLightPos;
};

struct miscBuf {
    float uTime;
    int uMode;
};

layout( std140, set = 0, binding = 0 ) uniform matBuf Matrices;
layout( std140, set = 1, binding = 0 ) uniform lightBuf Light;
layout( std140, set = 2, binding = 0 ) uniform miscBuf Misc;
layout( set = 3, binding = 0 ) uniform sampler2D uSampler;
```

* *binary large object*
VkResult
Init13DescriptorSetPool()
{
    VkResult result;
    VkDescriptorPoolSize vdpss[4];
    vdps[0].type = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
    vdps[0].descriptorCount = 1;
    vdps[1].type = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
    vdps[1].descriptorCount = 1;
    vdps[2].type = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
    vdps[2].descriptorCount = 1;
    vdps[3].type = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER;
    vdps[3].descriptorCount = 1;
    #ifdef CHOICES
    #endif
    VkDescriptorPoolCreateInfo vdpc;
    vdpc.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO;
    vdpc.pNext = nullptr;
    vdpc.flags = 0;
    vdpc.maxSets = 4;
    vdpc.poolSizeCount = 4;
    vdpc.pPoolSizes = &vdps[0];
    result = vkCreateDescriptorPool(LogicalDevice, IN &vdpci, PALLOCATOR, OUT &DescriptorPool);
    return result;
}

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.

MatrixSet DS Layout Binding: LightSet DS Layout Binding: MiscSet DS Layout Binding: TexSamplerSet DS Layout Binding:

- MatrixSet: binding = 0
- LightSet: binding = 1
- MiscSet: binding = 2
- TexSamplerSet: binding = 3
Step 2: Define the Descriptor Set Layouts

MatrixSet DS Layout Binding:  
- binding: 0  
- descriptorType:  
- descriptorCount: 1  
- pipeline stage(s): VK_SHADER_STAGE_VERTEX_BIT

LightSet DS Layout Binding:  
- binding: 0  
- descriptorType:  
- descriptorCount: 1  
- pipeline stage(s): VK_SHADER_STAGE_VERTEX_BIT

MiscSet DS Layout Binding:  
- binding: 0  
- descriptorType:  
- descriptorCount: 1  
- pipeline stage(s): VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT

TexSamplerSet DS Layout Binding:  
- binding: 0  
- descriptorType: VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER  
- descriptorCount: 1  
- pipeline stage(s): VK_SHADER_STAGE_FRAGMENT_BIT

unifom sampler2D uSampler;  
vec4 rgba = texture(uSampler, vST);
Step 3: Include the Descriptor Set Layouts in a Graphics Pipeline Layout

```
VkResult
Init14GraphicsPipelineLayout()
{
    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, IN &vplci, PALLOCATOR, OUT &GraphicsPipelineLayout );

    return result;
}
```
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

- **VkDescriptorBufferInfo** `vdbi0`:
  - `vdbi0.buffer = MyMatrixUniformBuffer.buffer;`
  - `vdbi0.offset = 0;`
  - `vdbi0.range = sizeof(Matrices);`

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

- ** VkDescriptorBufferInfo ** `vdbi1`:
  - `vdbi1.buffer = MyLightUniformBuffer.buffer;`
  - `vdbi1.offset = 0;`
  - `vdbi1.range = sizeof(Light);`

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

- **VkDescriptorBufferInfo** `vdbi2`:
  - `vdbi2.buffer = MyMiscUniformBuffer.buffer;`
  - `vdbi2.offset = 0;`
  - `vdbi2.range = sizeof(Misc);`

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

- ** VkDescriptorImageInfo ** `vdii0`:
  - `vdii.sampler = MyPuppyTexture.texSampler;`
  - `vdii.imageView = MyPuppyTexture.texImageView;`
  - `vdii.imageLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;`

  This struct identifies what texture sampler and image view it owns.

- ** 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;`

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

- ** 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.
Step 5: Tell the Descriptor Sets where their data is

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

```cpp
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 = &vdbi2;
vwds2.pImageInfo = (VkDescriptorImageInfo *)nullptr;
vwds2.pTexelBufferView = (VkBufferView *)nullptr;
```

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

```cpp
// 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_COMBINED_IMAGE_SAMPLER;
vwds3.pImageInfo = &vdii0;
vwds3.pTexelBufferView = (VkBufferView *)nullptr;
```

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

This struct links the pipeline to the image it is pointing to:

```cpp
VkGraphicsPipelineCreateInfo vgpci;
vgpci.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
vgpci.pNext = nullptr;
vgpci.flags = 0;
#if defined CHOICES
VK_PIPELINE_CREATE_DISABLE_OPTIMIZATION_BIT
VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT
VK_PIPELINE_CREATE_DERIVATIVE_BIT
#endif
vgpci.stageCount = 2;                           // number of stages in this pipeline
vgpci.pStages = vpssi;
vpci.pVertexInputState = &vpvissi;
vpci.pInputAssemblyState = &vpiasssi;
vpci.pTessellationState = (VkPipelineTessellationStateCreateInfo *)nullptr;
vpci.pViewportState = &vpvsssi;
vpci.pRasterizationState = &vprsssi;
vpci.pMultisampleState = &vpmsssi;
vpci.pDepthStencilState = &vpdsssi;
vpci.pColorBlendState = &vpccssi;
vpci.pDynamicState = &vpdssti;
vpci.layout = IN GraphicsPipelineLayout;
vpci.renderPass = IN RenderPass;
vpci.subpass = 0;                               // subpass number
vgpci.basePipelineHandle = (VkPipeline) VK_NULL_HANDLE;
vpci.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,
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.