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?

```cpp
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;
```
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.

```c
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:

```c
// 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;
```
**Descriptor Sets**

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

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

**GPU:**
- Uniform data used in the shader
  - Knows the shader data structure
  - Doesn't know where each piece of data starts

```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;
};
```

---

**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.
VkResult
Init13DescriptorSetPool()
{
    VkResult result;
    VkDescriptorPoolSize vdp[4];
    vdp[0].type = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
    vdp[0].descriptorCount = 1;
    vdp[1].type = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
    vdp[1].descriptorCount = 1;
    vdp[2].type = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
    vdp[2].descriptorCount = 1;
    vdp[3].type = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER;
    vdp[3].descriptorCount = 1;
#ifdef CHOICES
    VK_DESCRIPTOR_TYPE_SAMPLER
    VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE
    VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER
    VK_DESCRIPTOR_TYPE_STORAGE_IMAGE
    VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER
    VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER
    VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER
    VK_DESCRIPTOR_TYPE_STORAGE_BUFFER
    VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC
    VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC
    VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT
#endif
    VkDescriptorPoolCreateInfo vdpci;
    vdpci.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO;
    vdpci.pNext = nullptr;
    vdpci.flags = 0;
    vdpci.maxSets = 4;
    vdpci.poolSizeCount = 4;
    vdpci.pPoolSizes = &vdp[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.
 VkResult
Init13DescriptorSetLayouts()
{
VkResult result;
// DS #0:
VkDescriptorSetLayoutBinding MatrixSet[1];
MatrixSet[0].binding = 0;
MatrixSet[0].descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
MatrixSet[0].descriptorCount = 1;
MatrixSet[0].stageFlags = VK_SHADER_STAGE_VERTEX_BIT;
MatrixSet[0].pImmutableSamplers = (VkSampler *)nullptr;

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

// DS #2:
VkDescriptorSetLayoutBinding MiscSet[1];
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:
VkDescriptorSetLayoutBinding TexSamplerSet[1];
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;

uniform sampler2D uSampler;
vec4 rgba = texture(uSampler, vST);

---

### Step 2: Define the Descriptor Set Layouts

**MatrixSet DS Layout Binding:**
- binding: number of that type
- descriptorType: pipeline stage(s)
- descriptorCount: pipeline stage(s)

**LightSet DS Layout Binding:**
- binding: number of that type
- descriptorType: pipeline stage(s)
- descriptorCount: pipeline stage(s)

**MiscSet DS Layout Binding:**
- binding: number of that type
- descriptorType: pipeline stage(s)
- descriptorCount: pipeline stage(s)

**TexSamplerSet DS Layout Binding:**
- binding: number of that type
- descriptorType: pipeline stage(s)
- descriptorCount: pipeline stage(s)

---

**Array of Descriptor Set Layouts**

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

```c
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, &vplci, PALLOCATOR, OUT &GraphicsPipelineLayout );

    return result;
}
```
Step 4: Allocating the Memory for Descriptor Sets

vkAllocateDescriptorSets( )

---

```
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);

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

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

VkDescriptorImageInfo vdii0;
    vdii0.sampler = MyPuppyTexture.texSampler;
    vdii0.imageView = MyPuppyTexture.texImageView;
    vdii0.imageLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
```

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

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

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

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

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

```
VkWriteDescriptorSet vwds0;
    // ds 0:
    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;

    // 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 = &vdbi1;
    vwds1.pImageInfo = (VkDescriptorImageInfo *)nullptr;
    vwds1.pTexelBufferView = (VkBufferView *)nullptr;
```

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

This struct links a Descriptor Set to the buffer it is pointing to.
Step 5: Tell the Descriptor Sets where their data is

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

    // 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.pBufferInfo = (VkDescriptorBufferInfo *)nullptr;
    vwds3.pImageInfo = IN &vdii0;
    vwds3.pTexelBufferView = (VkBufferView *)nullptr;

    uint32_t copyCount = 0;
    // this could have been done with one call and an array of VkWriteDescriptorSets:
    vkUpdateDescriptorSets( LogicalDevice, 1, IN &vwds0, IN copyCount, (VkCopyDescriptorSet *)nullptr);
    vkUpdateDescriptorSets( LogicalDevice, 1, IN &vwds1, IN copyCount, (VkCopyDescriptorSet *)nullptr);
    vkUpdateDescriptorSets( LogicalDevice, 1, IN &vwds2, IN copyCount, (VkCopyDescriptorSet *)nullptr);
    vkUpdateDescriptorSets( LogicalDevice, 1, IN &vwds3, IN copyCount, (VkCopyDescriptorSet *)nullptr);
```

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

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

```cpp
    VkGraphicsPipelineCreateInfo vgpici;
    vgpici.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
    vgpici.pNext = nullptr;
    vgpici.flags = 0;
    #ifdef CHOICES
        VK_PIPELINE_CREATE_DISABLE_OPTIMIZATION_BIT
        VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT
        VK_PIPELINE_CREATE_DERIVATIVE_BIT
    #endif
    vgpici.stageCount = 2;                           // number of stages in this pipeline
    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;
    vgpici.subpass = 0; // subpass number
    vgpici.basePipelineHandle = (VkPipeline) VK_NULL_HANDLE;
    vgpici.basePipelineIndex = 0;

    result = vkCreateGraphicsPipelines( LogicalDevice, VK_NULL_HANDLE, 1, IN &vgpici, PALLOCATOR, OUT &GraphicsPipeline );
```

This struct links a Descriptor Set to the image it is pointing to.
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 );
```