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.

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

Descriptor Sets

Our example will assume the following shader uniform variables:

```glsl
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's data structure
- Knows where the data starts
- Knows the data size
- Doesn't know where each piece of data starts

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

- GPU:
  - Set 0
  - Binding 0
  - Uniform matrix data

- GPU:
  - Set 1
  - Binding 0
  - Light data

- GPU:
  - Set 2
  - Binding 0
  - Miscellaneous data

- GPU:
  - Set 3
  - Binding 0
  - Texture sampler data

### Uniform data created in a C++ data structure

```
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 vdps[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;
  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 = &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.

```
layout( std140, set = 0, binding = 0 ) uniform matBuf {
  mat4 uModelMatrix;
  mat4 uViewMatrix;
  mat4 uProjectionMatrix;
  mat3 uNormalMatrix;
};
```

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

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

```
layout( std140, set = 3, binding = 0 ) uniform sampler2D uSampler;
```
Step 2: Define the Descriptor Set Layouts

Matrix Set DS Layout Binding: 0
Light Set DS Layout Binding: 1
Misc Set DS Layout Binding: 2
Tex Sampler Set DS Layout Binding: 3

// DS #0:
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_VERTEX_BIT | 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
initGraphicsPipelineLayout();

VkResult
vkCreatePipelineLayout(LogicalDevice, INAndView, PPACKAGE, OUT &GraphicsPipelineLayout);

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

Computer Graphics
mjb – January 2, 2020

Computer Graphics
mjb – January 2, 2020
Step 4: Allocating the Memory for Descriptor Sets

```cpp
// 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 4: Allocating the Memory for Descriptor Sets

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

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

```cpp
// This struct identifies what buffer it owns and how big it 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);

// This struct identifies what texture sampler and image view it owns
VkDescriptorImageInfo vdii0;
    vdii0.sampler = MyPuppyTexture.texSampler;
    vdii0.imageView = MyPuppyTexture.texImageView;
    vdii0.imageLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;

// This struct links a Descriptor Set to the buffer it is pointing to
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;
    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 = IN &vdbi1;
    vwds1.pImageInfo = (VkDescriptorImageInfo *)nullptr;
    vwds1.pTexelBufferView = (VkBufferView *)nullptr;
```

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

```cpp
// This struct identifies what buffer it owns and how big it 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);

// This struct identifies what texture sampler and image view it owns
VkDescriptorImageInfo vdii0;
    vdii0.sampler = MyPuppyTexture.texSampler;
    vdii0.imageView = MyPuppyTexture.texImageView;
    vdii0.imageLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;

// This struct links a Descriptor Set to the buffer it is pointing to
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;
    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 = IN &vdbi1;
    vwds1.pImageInfo = (VkDescriptorImageInfo *)nullptr;
    vwds1.pTexelBufferView = (VkBufferView *)nullptr;
```
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 = {
    .sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET,
    .pNext = nullptr,
    .dstSet = DescriptorSets[2],
    .dstBinding = 0,
    .dstArrayElement = 0,
    .descriptorCount = 1,
    .descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
    .pBufferInfo = &vdbi2,
    .pImageInfo = nullptr,
    .pTexelBufferView = nullptr,
};
```

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

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

```cpp
VkWriteDescriptorSet vwds3 = {
    .sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET,
    .pNext = nullptr,
    .dstSet = DescriptorSets[3],
    .dstBinding = 0,
    .dstArrayElement = 0,
    .descriptorCount = 1,
    .descriptorType = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER,
    .pBufferInfo = nullptr,
    .pImageInfo = &vdii0,
    .pTexelBufferView = nullptr,
};
```

### Sidebar: The Entire Collection of Descriptor Set Paths

- `vkCreateDescriptorSets()`: Allocate memory for particular Descriptor Sets
- `vkCreateBufferInfo()`: Tell a particular Descriptor Set where its CPU data is
- `vkCreateImageInfo()`: Describe a particular Descriptor Set layout and use it in a specific Pipeline layout
- `vkCreateSetLayoutBinding()`: Describe a particular Descriptor Set layout
- `vkCreateSetLayoutCreateInfo()`: Create the pool of Descriptor Sets for future use
- `vkCreateCommandSets()`: Make a particular Descriptor Set "current" for rendering
- `vkCmdBindDescriptorSets()`: Bind Descriptor Sets into the Command Buffer when Drawing
- `vkUpdateDescriptorSets()`: Re-write CPU data into a particular Descriptor Set
- `vkCmdBindPipeline()`: Set the Pipeline the Command Buffer will use
- `vkCmdBindPipelineSets()`: Bind the Pipeline to the Command Buffer
Sidebar: Why Do Descriptor Sets Need to Provide Layout Information to the Pipeline Data Structure?

The pieces of the Pipeline Data Structure are fixed in size – with the exception of the Descriptor Sets and the Push Constants. Each of these two can be any size, depending on what you allocate for them. So, the Pipeline Data Structure needs to know how these two are configured before it can set its own total layout.

Think of the DS layout as being a particular-sized hole in the Pipeline Data Structure. Any data you have that matches this hole’s shape and size can be plugged in there.

The Pipeline Data Structure

- Fixed Pipeline Elements
- Specific Descriptor Set Layout